UNIVERSIDAD CARLOS III DE MADRID

ESCUELA POLITÉCNICA SUPERIOR

Framework de autenticación único

PROYECTO FIN DE CARRERA

INGENIERÍA SUPERIOR DE TELECOMUNICACIÓN

Autor: Diego Escalada López
Tutor: Julio Villena Román

Octubre 2009

Título: Framework de autenticación único

Autor: Diego Escalada López

Tutor: Julio Villena Román

EL TRIBUNAL

Presidente: Carlos Jesús Bernardos Cano

Secretario: Pablo Basanta Val

Vocal: Rui Santos

Realizado el acto de defensa del Proyecto Fin de Carrera el día 07 de Octubre de 2009 en Leganés, en la Escuela Politécnica Superior de la Universidad Carlos III de Madrid, acuerda otorgarle la CALIFICACIÓN de:

Fdo: Presidente Fdo: Secretario Fdo: Vocal

Agradecimientos

A mi hermana, porque es la que siempre esta y siempre estará a mi lado. A mis padres porque esto les hace más feliz a ellos casi que a mí. A mis abuelas, porque son geniales cada una a su manera. A Juanlu y Anastasia porque sin ellos no estaría escribiendo esto y porque espero que sigan formando parte de mi vida en el futuro. A Jorge porque él me enseño a poner un sofá en el sitio indicado y sentirme cómodo en las situaciones complicadas y porque es con él con quien acabaré mis días. A Pedro, Sergio y Juan Pablo porque solo espero que la vida les trate bien y que yo este para ellos como ellos están para mí. A Álvaro, el capitán, que me enseño lo importante de la motivación y el esfuerzo para conseguir los sueños. A Dani y a Miguel porque verles tan felices me hace feliz a mí también, casi siempre. A Oscar, mi supercolega. A Raúl por compartir tantas mañanas en Motorola y por ensenarme tanto de la confianza en uno mismo. A Ana Bea por su sonrisa, entre otras muchas cosas. A Diana por ser tan empanada y genial. Al resto de compañeros de universidad por hacerme disfrutar esa etapa de mi vida. A la Universidad en la que tantas y tantas cosas he aprendido. A Alber porque sé que va a ser el compañero de piso perfecto. A Carmelo porque algún día descubrirá lo especial que es, tanto para mí como para el mismo. A Jorge, Canario, Dani, Victor y Pearson por las conversaciones sobre la vida que nos quedan pendientes. A BEST por mostrarme el camino. A Reme por ayudarme a comenzarlo y estar ahí cuando se me hace cuesta arriba. A Silvia por hacerme sentir normal en las situaciones en las que la vida no te lo hace sentir así. A Ceci, la mejor, sin duda alguna. A Chumi, Saquitos, Dudu, Argui y demás gente de mi pueblo por hacerme reír tantas y tantas veces. A Richi y a Quique porque con ellos comencé a ser lo que ahora soy. Al Kimi porque aunque no debamos mantener conversaciones sobre la vida nos encanta desafiar los limites. A Julio por ayudarme con este proyecto. A los errores que he cometido y cometeré porque de ellos he aprendido y aprenderé las cosas más importantes de la vida.

Resumen

El proyecto surge tras una necesidad de implantar un sistema de obtención de credenciales en la red corporativa de una gran empresa cliente de Oesia[1], antiguo IT Deusto, que por motivos de confidencialidad no se cita en el texto de un documento público. Una vez propuesta la idea el proyecto se realizó dentro de la empresa Novatia Soluciones Tecnológicas, una pequeña empresa dedicada al desarrollo de aplicaciones y la consultoría tecnológica. El proyecto se le bautizó como PAO o *Portal de Acceso Oesia*.

El proyecto cubre todas las funciones de toma de requisitos, análisis, diseño de la infraestructura de red y desarrollo del software.

El esquema general consiste en la implantación de un sistema que permitiese a cualquier usuario de la red autenticarse de manera automática en cualquiera de las aplicaciones corporativas con la simple inserción de un dispositivo de almacenamiento seguro en el ordenador[2].

Para ello se desarrollan una interfaz sobre el dispositivo de almacenamiento seguro de claves de Aladdin, un servicio Web encargado de la interactuación contra el servidor de claves y una interfaz que puede ser utilizada por aplicaciones de terceros que interactúa tanto con la primera de las interfaces como con el servicio Web desarrollado. Todos estos componentes junto con un servidor de directorio de Microsoft, que almacena las claves de sesión de los diferentes usuarios en las diferentes aplicaciones, forman el denominado Framework de autenticación única.

[1] http://www.oesia.com

[2] El dispositivo de almacenamiento seguro utilizado en este proyecto es el Aladdin eToken PRO 32K 4.28. Más referencias en http://www.aladdin.es

Índice

Índice de figuras

Índice de tablas

Acrónimos

A continuación se muestran los acrónimos que se utilizan en el documento:

ADAM	Active Directory Application Mode
USB	Universal Serial Bus
TLS	Transport Layer Security
SSL	Secure Socket Layer
SSH	Secure Shell
SO	Security Officer
S/MIME	Secure / Multipurpose Internet Mail Extensions
PIN	Personal Identification Number
PGP	Pretty Good Privacy
NIST	National Institute of Standards and Technology
MD5	Message Digest Algorithm 5
IPsec	Internet Protocol Security
ECMA	European Computer Manufactures Association
DBMS	Database Management Server
CAU	Centro Atención de Usuario

1 Introducción

1.1 Motivación

El proyecto no es innovador en el sentido tecnológico ya que todas las tecnologías utilizadas durante su desarrollo son tecnologías ya estudiadas y de uso común en otras aplicaciones del mercado. Sin embargo, una vez que el departamento de seguridad decide que ninguna de las aplicaciones del mercado se ajusta totalmente a los requisitos funcionales que demandaban y se decantan por el desarrollo íntegro de una solución nueva, el proyecto cobra unas nuevas dimensiones.

La realización de un análisis íntegro del sistema de seguridad de una empresa es un reto importante. Además el proyecto incluye el desarrollo de la aplicación de bajo nivel para el dispositivo de almacenamiento de claves y también el desarrollo de un servicio web para intercomunicar los clientes con el servidor de claves que se desarrollaría posteriormente.

De esta manera se expresa el hecho de que el proyecto es ambicioso, en el sentido de proyección, ya que será utilizado en una red corporativa considerable, pero abarcable para un analista programador o desarrollador con el tiempo necesario. El proyecto tardó en finalizarse siete meses y tuvo un retraso aproximado de un mes y medio debido a las pruebas en cliente y la instalación de los elementos finales en la red corporativa de la empresa; Fueron necesarios trámites, auditorías y validaciones finales de todos los departamentos involucrados como requisito previo a la puesta en producción de un proyecto que afecta a todos los usuarios en su trabajo diario.

En último término, la motivación principal del proyecto se debe a la necesidad de facilitar el trabajo a todas las personas. La red de la empresa final es utilizada por un porcentaje mayor de usuarios con accesibilidad limitada que el de otras compañías ya que su razón social se basa en la integración de dichas personas al mercado laboral, por lo que la sencillez y mejora de los procesos que cada usuario puede realizar a diario constituía a la vez una motivación y un punto clave en el diseño.

1.2 Descripción del problema

El objetivo principal es implantar un sistema que permitiese la obtención de credenciales con la mayor sencillez de uso que se pudiese conseguir. Para la consecución de dicho objetivo se partía con dos requisitos previos que la solución debe proporcionar. En primer lugar los administradores de sistema de la empresa, como en muchas de las grandes empresas, están subcontratados a terceros y el departamento de seguridad no confiaba en la figura del administrador de forma plena. En segundo lugar se debe tener en cuenta que muchos usuarios pueden olvidar el dispositivo y el sistema debe estar preparado para clonar el dispositivo y las datos que éste contenía, de forma que el usuario pueda trabajar como si ese hecho no hubiese ocurrido y sin que se produzca un colapso del departamento de atención al usuario.

Ambos requisitos utilizados de manera conjunta excluían las soluciones comerciales en las que los administradores podían en cualquier momento conocer cualquier dato del sistema y en las que si el usuario administrador únicamente podía crear una nueva contraseña dentro de la red, sobrescribiendo la anterior, sin conocerla con anterioridad.

La aplicación debía además proveer los servicios en caso de que el servidor de claves no estuviese disponible por cualquier motivo. En el momento del comienzo del desarrollo la empresa disponía de un servidor ADAM (ver apartado 2.1.4), servidor LDAP de Microsoft, donde cada se guarda como texto en claro el par usuario y contraseña que cada usuario de la red tenía en las distintas aplicaciones publicadas de manera corporativa. De esta manera los usuarios de las aplicaciones corporativas no necesitaban recordar muchas contraseñas diferentes para cada aplicación, sino únicamente su contraseña de acceso a la red.

Este hecho ya puede considerarse como no aceptable en ciertas compañías, pero en esta se estaba convirtiendo en una importante brecha de seguridad ya que los administradores externos conocían la contraseña de cualquier usuario en cualquier aplicación corporativa, incluyendo jefes de departamento con altas responsabilidades estratégicas y financieras dentro de la compañía.

Llegados a este punto parece obvio pensar que la solución pasase por un sistema común de autenticación para todas las aplicaciones corporativas soluciona el problema. Evidentemente, así es, pero la gran mayoría de los sistemas de la compañía que trabajan con los datos financieros, bancarios, etc. son sistemas antiguos que no ofrecen posibilidad de autenticación externa, siendo alguno de ellos incluso desarrollos propietarios por empresas actualmente desaparecidas o productos comerciales que se encuentran fuera de soporte en la actualidad.

El sistema final pretendía en principio almacenar en el mismo servidor ADAM todos los pares de usuarios y contraseñas de todos los usuarios de manera segura, permitiendo la lectura a los administradores de sistemas, como hasta el momento, pero dicha información ya no será relevante para dichos administradores que tendrán acceso a una información cifrada que únicamente cada usuario de la red corporativa podrá descifrar. De esta manera queda totalmente solventado el punto en el que la empresa quería no permitir a los administradores de sistema el conocimiento de la información, pero continuando con la delegación de la administración de dichos servidores. Para el cifrado y descifrado de datos se optó como mejor solución una técnica asimétrica a partir de un par de claves pública-privada propia de cada uno de los usuarios de la red.

La importancia de que los usuarios pudiesen recuperar su par de claves en cualquier momento radicaba en que una vez que toda la información ha sido cifrada, con las tecnologías que se van a exponer a continuación en la memoria, si el usuario no tiene acceso a su par de claves todas las contraseñas de todos los sistemas tienen que ser creadas de nuevo.

Para facilitar el acceso al par de claves por los usuarios se pretendía distribuir unos dispositivos USB que contuviesen dicha información, pero ¿qué pasaba en el caso de que el usuario se olvidase del dispositivo o lo perdiese? Tenía que crearse de nuevo exactamente con la misma información que contuviese el anterior ya que sino el usuario perdía de nuevo el acceso a las aplicaciones. Si se le asignase un nuevo par de claves de usuario, distinto al original, se tendría que re-cifrar todos los usuarios y contraseñas de todas las aplicaciones a las que tenía acceso dicho usuario y crear las nuevas contraseñas en cada uno de los sistemas. Dicha creación de nuevas contraseñas en algunos casos conllevaba un tiempo de replicación durante el que el usuario no tenía acceso a las aplicaciones. Este tiempo podía ser desde unas horas hasta, en el peor caso, una jornada de trabajo lo que suponía un gran gasto para la compañía.

Una vez identificado como probable el hecho de que a un usuario se le olvide su dispositivo y considerando el gran tamaño de la empresa se temía hipotecar el trabajo del departamento de sistemas a la mera tarea de establecer de nuevo par de claves que impactase en la replicación de las nuevas claves de todos los sistemas o a la inicialización de nuevos dispositivos, por ello se consideraba como crítico en la solución final la inclusión de un sistema que permitiese recuperar la información del dispositivo de usuario de manera rápida y segura.

El dispositivo ofrecía dos ventajas considerables y una desventaja. Entre las ventajas se desatacaban el almacenamiento seguro del par de claves que quedaban protegidos por un PIN o contraseña y el hecho de que únicamente el poseedor del dispositivo tenía acceso a las aplicaciones y ya no se podía delegar una actividad diciendo a un compañero o subordinado el usuario y contraseña de la aplicación ya que sin el dispositivo se perdía el acceso a todas las aplicaciones corporativas. Como desventaja aparecía el coste elevado que suponía la distribución de dicho dispositivo a todos los usuarios de la red. Sin embargo, debido al tamaño de la compañía, ésta había acordado un precio razonable con el fabricante de los dispositivos de manera que el coste ha resultado asumible y no supone una limitación en la solución final.

1.3 Objetivos de diseño

Los objetivos de la aplicación son varios todos ellos igual de importantes porque impactan de manera directa en el negocio de la compañía. Cada uno de estos objetivos se comprobará para ver si el proyecto tiene éxito.

- Desarrollo del software en .NET ya que es el estándar de desarrollo en la compañía del cliente final. Del mismo modo ADAM es el servidor LDAP que se utiliza dentro de la compañía, por lo que aunque la aplicación debe ser diseñada de manera general, todas las pruebas realizadas serán sobre ese entorno.
- Despliegue rápido y escalado en la red de la compañía. No se podía ni pretendía distribuir el dispositivo de seguridad a todos los usuarios ya que ello suponía una labor logística muy considerable. Por tanto, durante un período de tiempo que puede llegar a ser ilimitado, la nueva infraestructura de obtención de los credenciales debe coexistir con la infraestructura anterior de manera que ningún usuario se vea afectado.
- Conseguir que la empresa superase satisfactoriamente cualquier auditoría de seguridad. Este punto es especialmente importante porque la empresa no se puede permitir fallos o agujeros en la seguridad como los que existían previos a la implantación de la nueva arquitectura ya que repercutían directamente sobre la imagen del departamento de seguridad al resto de la compañía y, esto obviamente, en el presupuesto que se le asignaba anualmente.
- Fiabilidad de la nueva infraestructura creada. La nueva red no podía contener errores ya que los usuarios dejaban de tener acceso a las aplicaciones de la empresa y, en este caso, los costes resultaban muy elevados.
- Disponibilidad de los servicios de recuperación del dispositivo. Cualquier usuario con problemas de extravío u olvido de la contraseña debe ser atendido de manera rápida y eficaz, reduciendo al mínimo el tiempo en el que no puede realizar su trabajo.

1.4 Esquema del documento

La memoria de este Proyecto Fin de Carrera comienza con el capítulo de introducción, donde se ha definido la motivación, el problema y los objetivos. El segundo capítulo recoge los antecedentes al estudio y los conceptos teóricos de los que se ha hecho uso.

En un tercer capítulo se explicará el diseño lógico de la arquitectura, las diferentes aplicaciones que toman parte y el análisis de cada una de ellas. Además este capítulo incluirá el diagrama de clases de dichas aplicaciones y una descripción de las funciones principales y la configuración de los servidores para la instalación de dicha aplicación.

En el cuarto se expondrán las pruebas concretas que se han realizado sobre la arquitectura, tanto de rendimiento como funcionales y el resultado a cada una de las mismas.

En el quinto capítulo se encuentran las conclusiones finales y se plantearán futuras continuaciones del estudio descrito en esta memoria.

Por último, la bibliografía se incluye al final del documento.

2 Estado del arte

En el siguiente apartado se presentan de manera resumida las tecnologías y protocolos implicados en el desarrollo de este proyecto. En concreto este capítulo se centra en introducir las conceptos fundamentales sobre el entorno en el que se ha realizado el desarrollo de software como .NET Framework y C#, en una descripción breve del protocolo LDAP y de como Microsoft implementa dicho protocolo y ofrece al mercado su servicio de directorio bajo el nombre de ADAM.

Adicionalmente se detallan las bases matemáticas sobre las que se basan los algoritmos de seguridad utilizados en el desarrollo, así como el propósito de uso dentro del proyecto. En concreto se detalla el estándar de conexión a dispositivos de almacenamiento seguro, PKCS#11, el estándar de cifrado asimétrico para el almacenamiento seguro, el estándar de cifrado simétrico que busca un compromiso entre velocidad y seguridad y el estándar de hash utilizado en el proyecto para la comprobación y generación de claves aleatorias generadas.

2.1 Lenguajes y entornos de desarrollo

2.1.1 .NET Framework 2.0

.NET Framework es un componente integral de Windows que soporta el desarrollo y la ejecución de las aplicaciones y los servicios Web XML. Los principales objetivos de .NET Framework se enumeran a continuación (1):

- Proveer un entorno orientado a objetos consistente donde el código sea almacenado y ejecutado localmente, pero distribuido a través de Internet o ejecutado de manera remota.
- Proveer un entorno que minimice el desarrollo de software y los conflictos de versiones durante el proceso.
- Proveer un entorno de ejecución que promueva la ejecución segura del código, incluyendo el código creado por terceros anónimos o de confianza limitada.
- Proveer un entorno para el desarrollo de código que elimine los problemas de rendimiento que tienen los lenguajes basados en script o los entornos interpretados.
- Hacer la experiencia del desarrollador consistente a través de los variados tipos de aplicaciones que se pueden desarrollar, como aplicaciones basadas en Windows o en Web.
- Construir toda la comunicación con los estándares de la industria para asegurar que cualquier código desarrollado en .NET Framework puede integrarse con cualquier otro código.

.NET Framework tiene dos componentes fundamentales: el CLR (*Common Language Runtime*) y el FCL (*Framework Class Library*).

El CLR es la base sobre la que se apoya el resto del entorno. El CLR cumple las funciones de administrar el código durante la ejecución, proveyendo servicios base como la gestión de la memoria, la gestión de los hilos de ejecución y gestión remota, mientras se encarga de la seguridad de tipos y otros tipos de mejora de código nativo que impacten directamente en el incremento de la seguridad y robustez del código que se está ejecutando.

El concepto de código administrado es muy importante es este entorno y es el principio fundamental en el que se basa. El código que se ejecuta en el CLR se le conoce como código

administrado y el código que se ejecuta utilizando otros servicios del sistema operativo se le conoce como código no administrado.

El otro componente principal de la arquitectura .NET es el FCL que se define como una colección exhaustiva orientada a objetos de tipos que los desarrolladores pueden utilizar para desarrollar aplicaciones que pueden variar desde las tradicionales por línea de comando o aplicaciones que utilicen la interfaz de usuarios a aplicaciones que incorporen las últimas innovaciones como los formularios Web y los servicios Web XML (1).

El FCL y el CLR es la base del resto de servicios de alto nivel que provee el .NET Framework. Una visión global del entorno puede observarse en la siguiente figura (2):

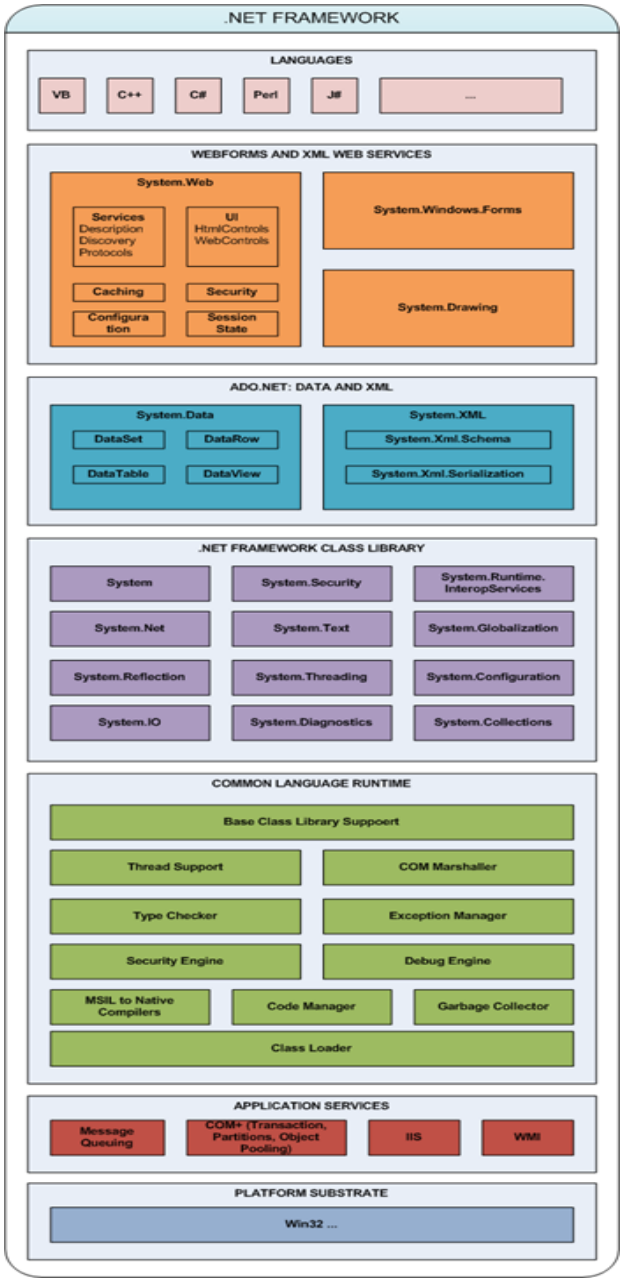


Figura 1 .NET Framework

En el momento de inicio del proyecto la versión estable era .NET Framework 2.0. En la actualidad existen versiones superiores del Framework, 3.0 y 3.5, estando la versión 4.0 bajo desarrollo. La siguiente figura (3) muestra como la versión del CLR se ha mantenido en las nuevas versiones desarrolladas de manera que el código puede seguir ejecutándose sin modificación alguna sobre cualquiera de los entornos que el cliente disponga.

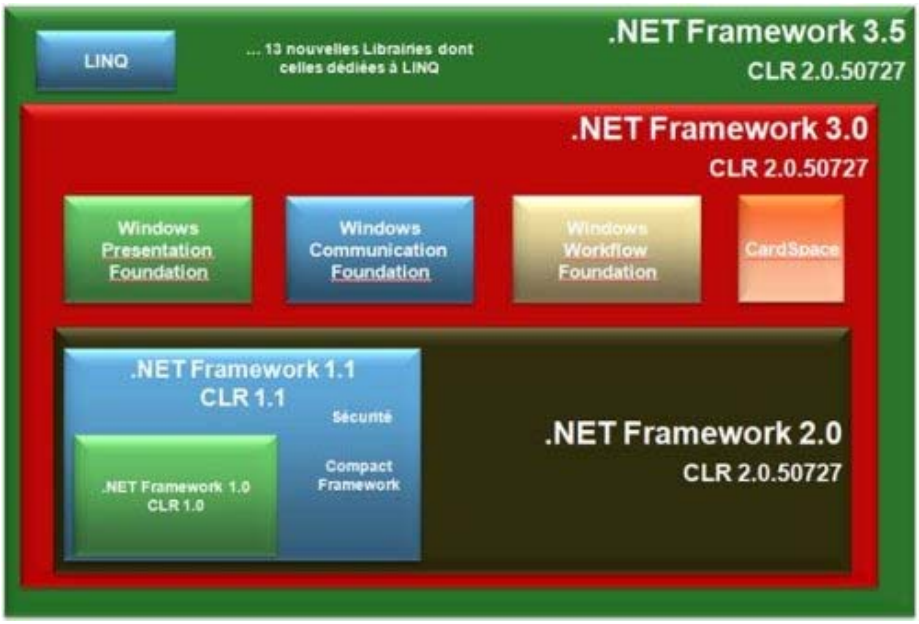


Figura 2 Versiones .NET Framework

2.1.2 C#

El CLI (*Common Language Infraestructure*) es una especificación desarrollada conjuntamente por Microsoft, Intel y Hewlett-Packard como un estándar ECMA. Dicha especificación define un entorno en el que varios lenguajes de alto nivel pueden ser utilizados en diferentes plataformas sin necesidad de reescribir código propio de la plataforma en donde se ejecuta (4). El CLR es una implementación específica del CLI.

La idea inicial de C# era proveer a la comunidad de desarrolladores con un lenguaje tan fácil de utilizar como Visual Basic, pero que tuviese la misma potencia y flexibilidad que tiene el C++. C# se basa fundamentalmente en C++ al que se le han añadido todas las funcionalidades necesarias para que fuese tan fácil de mantener como Visual Basic y se le han eliminado algunas de las funcionalidades más antiguas de dicho lenguaje, si bien el objetivo fundamental del diseño fue simplicidad y no potencia, C# cede una pequeña porción de rendimiento frente lenguajes como C y C++ frente a un código más estable y productivo en el largo plazo, proveyendo de las siguientes funcionalidades (5):

- Simplicidad
- Consistencia
- Modernidad
- Seguridad de tipos
- Orientado a objetos
- Flexibilidad
- Escalabilidad

- Compatibilidad
- Control de versiones

2.1.3 LDAP

De las siglas en inglés *Ligthweight Directory Access Protocol* o Protocolo Ligero de Acceso a Directorio como se ha adoptado al término castellanizado es un protocolo de aplicación basado en X.500 (6) (7), que permite el acceso a un servicio de directorio ordenado y distribuido para buscar diversa información en un entorno de red (8).

Aunque coloquialmente se confunde LDAP como una base de datos relacional debe especificarse claramente que un directorio LDAP es un tipo de base de datos no relacional fuertemente optimizado para la lectura que soporta cientos o miles de accesos por minuto.

La actual popularidad de LDAP es la culminación de un conjunto de factores. Tal vez la mayor ventaja de LDAP es la interoperabilidad ya que puede accederse al directorio LDAP desde casi cualquier plataforma de computación y desde cualquier aplicación debido a la diversidad de aplicaciones existentes y la facilidad de personalización para las aplicaciones internas que se desarrollen dentro de una red corporativa.

El protocolo LDAP es utilizable por distintas plataformas y basado en estándares, de ese modo las aplicaciones no necesitan preocuparse por el tipo de servidor en que se hospeda el directorio. De hecho, LDAP está encontrando una más amplia aceptación a causa de ese estatus como estándar de Internet. Un servidor LDAP puede ser cualquiera de entre una gran variedad de implementaciones abiertas, como openLDAP, o comercial, como Active Directory de Microsoft, eDirectory de Novell o iPlanet de Sun.

La mayoría de los servidores LDAP son simples de instalar, de mantener y de optimizar. Los servidores LDAP tienen integrados un sistema de réplica que potencia la escalabilidad del servicio de directorios en empresas distribuidas, permitiendo la transmisión parcial o total de los datos de cada una de las oficinas remotas.

LDAP permite controlar el acceso al directorio mediante autorizaciones personalizables utilizando ACLs (del inglés *Access Control List* o Lista de Control de Acceso en castellano). Las listas de control de acceso permiten controlar el acceso dependiendo de quién está solicitando los datos, qué datos están siendo solicitados, dónde están los datos almacenados, y otros aspectos del registro que está siendo modificado. Al integrar las listas de control de acceso dentro del directorio LDAP no es necesaria una lógica de aplicación dentro del nivel de usuario.

LDAP es particularmente utilizable para almacenar información que se desee leer desde muchas localizaciones, pero que no sea actualizada frecuentemente. Por ejemplo, ejemplos típicos de utilización del directorio LDAP dentro de las redes corporativas se citan a continuación:

- Servicios de autenticación dentro de la red corporativa
- Sistema de almacenamiento de certificados de usuario para el acceso seguro a los recursos
- Guía de información de los usuarios de la empresa
- Información de contacto de clientes externos

- Información de configuración para paquetes de software distribuidos

2.1.4 ADAM

ADAM (9) es un sistema propietario de Microsoft que provee servicios de directorio basados en LDAP definido originalmente como una versión más ligera del producto Active Directory integrado en las versiones de servidor de Microsoft desde Microsoft Windows 2000 Server.

Originalmente inspirado por la emergencia de productos basados en LDAP durante mediados de los años 90, un gran número de compañías buscaban soluciones de negocio para el desarrollo de aplicaciones basadas en directorio tales como la distribución de claves en NOS (del inglés *Network Operating System* o sistema operativo en red), integraciones con sistemas de PKI (del inglés *Public Key Infraestructure* o infraestructura de clave pública), sistemas de páginas blancas o amarillas, sistemas de acceso común a las diferentes plataformas de la compañía, etc.

El resultado fue que muchas compañías adoptaron un servicio de directorio diferente para cada una de las soluciones que se adoptaron dentro de la red corporativa. Existía un sistema de directorio para la distribución de claves en los NOS, uno diferente para el acceso remoto basado en infraestructuras de clave pública, uno diferente para la extranet y para el acceso Web, etc. Además era común en las organizaciones desarrollar múltiples servicios de directorio basados en diferentes tecnologías de directorio (9).

La pregunta que surge en ese momento es si todos los servicios de directorio estaban basados en LDAP, ¿por qué las compañías no eran capaces de estandarizar una única tecnología de directorio? La respuesta se encontraba con los siguientes obstáculos:

- Falta de interoperabilidad.
- Falta de opciones.
- Falta de coordinación.
- Falta de interoperabilidad en la seguridad.

ADAM es por tanto una capacidad de Active Directory que se dirige a ciertos escenarios relacionados con las aplicaciones que utilizan servicios de directorio. ADAM corre como un servicio que no pertenece al sistema operativo y, por tanto, no requiere que se instale en un controlador de dominio. Además ello permite que varias instancias de ADAM corran dentro de un mismo servidor y que cada una de ellas sea configurada de manera independiente.

ADAM es, por tanto, un directorio de servicios LDAP que aparece en el mercado encontrándose con los obstáculos mencionados con anterioridad, mantiene la flexibilidad y ayuda a la organización evitando incrementar los costes en infraestructuras.

Muchas aplicaciones requieren un servicio de directorio simple. De manera que la información que almacena puede que no sea de interés mundial o que no sea necesaria su replicación al resto de ramas de la compañía. La información puede requerir un diferente nivel de servicio que el nivel de servicio ofrecido por los controladores de dominio que hospedan un directorio NOS. Por ejemplo, hay aplicaciones que almacenan información volátil que causa un alto

tráfico de replicación y el atasco en los recursos de la red en caso de que fuesen almacenados en el directorio NOS.

2.2 Seguridad

En el siguiente apartado se van a comentar las diferentes tecnologías en el mundo de la seguridad que son de aplicación durante el desarrollo de este proyecto. Se definirán los esquemas concretos y arquitecturas utilizadas sin hacer una introducción general al mundo de la seguridad y la criptografía y detallando de una manera más precisa los procesos implicados cuando proceda.

A partir del conocimiento interno de las mismas se justificará en cada apartado y cuando corresponda el por qué de la elección de dicha tecnología de seguridad para el Framework y no otra de esa misma familia.

2.2.1 PKCS#11

PKCS, del inglés *Public-Key Cryptography Standards* traducido al castellano como estándares de criptografía de clave pública, es un conjunto de especificaciones elaboradas por RSA Laboratories (10) en cooperación con un selecto grupo mundial de desarrolladores con el propósito de acelerar el desarrollo de la criptografía de clave pública. Se publicó en primer lugar en 1991 (11) como resultado de las reuniones con un pequeño grupo de compañías que acogieron el desarrollo de la criptografía de clave pública. Los documentos PKCS se han convertido en elementos ampliamente referenciados e implementados. Las contribuciones realizadas desde PKCS se han convertido en una parte importante de algunos estándares de facto como los documentos ANSI X.509, PKIX, S/MIME y SSL.

Los estándares PKCS se pueden observar en la siguiente tabla (12). De entre todos el que aquí afecta es el dedicado a la interfaz con los dispositivos de seguridad, esto es PKCS#11:

Estándar	Versión	Nombre
PKCS#1	2.1	Estándar criptográfico RSA. RFC 3447
PKCS#2	-	Obsoleto. Sustituido por el PKCS#1
PKCS#3	1.4	Estándar de intercambio de claves Diffie-Hellman
PKCS#4	-	Obsoleto. Sustituido por el PKCS#1
PKCS#5	2.0	Estándar de cifrado basado en contraseñas. RFC 2898
PKCS#6	1.5	Estándar de sintaxis de certificados extendidos
PKCS#7	1.5	Estándar sobre la sintaxis del mensaje criptográfico. RFC 2315
PKCS#8	1.2	Estándar sobre la sintaxis de la información de clave privada
PKCS#9	2.0	Tipos de atributos seleccionados
PKCS#10	1.7	Estándar de solicitud de certificación. RFC 2986
PKCS#11	**2.20**	**Interfaz de dispositivo criptográfico ("Cryptographic Token Interface" o cryptoki)**
PKCS#12	1.0	Estándar de sintaxis de intercambio de información personal
PKCS#13	–	Estándar de criptografía de curva elíptica. En desarrollo
PKCS#14	–	Generación de número pseudo-aleatorios. En desarrollo
PKCS#15	1.1	Estándar de formato de información de dispositivo criptográfico

Tabla 1 PKCS

El estándar PKCS#11 especifica una API llamado Cryptoki para el almacenamiento de datos e información en los dispositivos y la realización de funciones criptográficas. Cryptoki es una abreviatura de **CRYP**tographic **TO**ken **I**nterface que sigue una filosofía orientada a objetos teniendo en cuenta la independencia de tecnología (cualquier tipo de dispositivo), el acceso simultáneo a los recursos (varias aplicaciones accediendo a varios dispositivos) y presentando a las aplicaciones una idea común y lógica denominada token criptográfico o, simplemente, token que es el objeto que representa al dispositivo.

El estándar describe los tipos de datos y funciones disponibles para una aplicación que requiera de servicios criptográficos desarrollados en ANSI C. Estos tipos de datos son proporcionados por RSA Laboratories en cada una de sus versiones para las diferentes plataformas. Existen otras implementaciones del estándar en otros lenguajes como .NET o Java, pero que no son mantenidos por la propia empresa.

Cryptoki aísla las aplicaciones de los detalles del dispositivo criptográfico. La aplicación es portable ya que no necesita realizar cambios para funcionar con otro dispositivo o en una plataforma diferente.

La versión actual de PKCS#11, como se ha comentado anteriormente, soporta diferentes funciones criptográficas como son algoritmos de clave simétrica, funciones hash y distribución de claves. Cada nueva versión de dicha especificación implementa los nuevos algoritmos y especifica y define los procesos necesarios para que los ensambladores de dispositivos puedan incluir sus propios mecanismos, aunque en este caso se pierda de vista la interoperabilidad de la aplicación desarrollada.

Cryptoki no especifica ningún mecanismo para la diferenciación entre diferentes usuarios, ya que se considera que un dispositivo de almacenamiento es de uso exclusivo, por lo que el modelo estima que el dispositivo incluye únicamente las claves de usuario y unos cuantos certificados como mucho. Si bien es posible que los dispositivos ofrezcan funcionalidades adicionales importantes no relacionadas con la criptografía, PKCS#11 no incluye ningún mecanismo que los integre.

2.2.1.1 Propósitos de diseño

Cryptoki fue desarrollado con el propósito de crear un estándar entre las aplicaciones y todos los tipos de dispositivos portátiles como smart cards, PCMCIA… todos los estándares existentes, de facto u oficiales, se aplican a estos dispositivos a otros niveles, como puede ser a nivel electrónico.

Lo que quedaba pendiente de ser definido eran los comandos particulares para realizar funciones criptográficas. No era factible definir diferentes comandos para cada tipo de dispositivo, y no solucionaría el problema general de conseguir una interfaz independiente del dispositivo. Por tanto, el principal objetivo de PKCS#11 fue crear una interfaz de bajo nivel que abstrajera los detalles de los dispositivos. El segundo de los objetivos era la posibilidad de que el dispositivo fuese accedido por más de una aplicación y que una misma aplicación pudiese acceder a más de un dispositivo.

Hay que comentar que PKCS#11 únicamente trata de complementar, pero no competir, con otros estándares emergentes como son GSS-API (del inglés *Generic Security Services API* o API para servicios de seguridad genéricos en castellano), RFC 2743 y 2744, y GCS-API (del inglés *Generic Cryptographic Service API* o API para servicios de criptografía) de la compañía X/Open.

2.2.1.2 *Modelo General*

El modelo general de la API se muestra en la siguiente figura. En su parte superior puede observarse un número indeterminado de aplicaciones que quieren acceder a los servicios criptográficos situados en la parte inferior de la figura. Cada dispositivo puede ejecutar una o varias operaciones al mismo tiempo, sin necesidad de que todas las operaciones deban estar asociadas a un usuario en concreto.

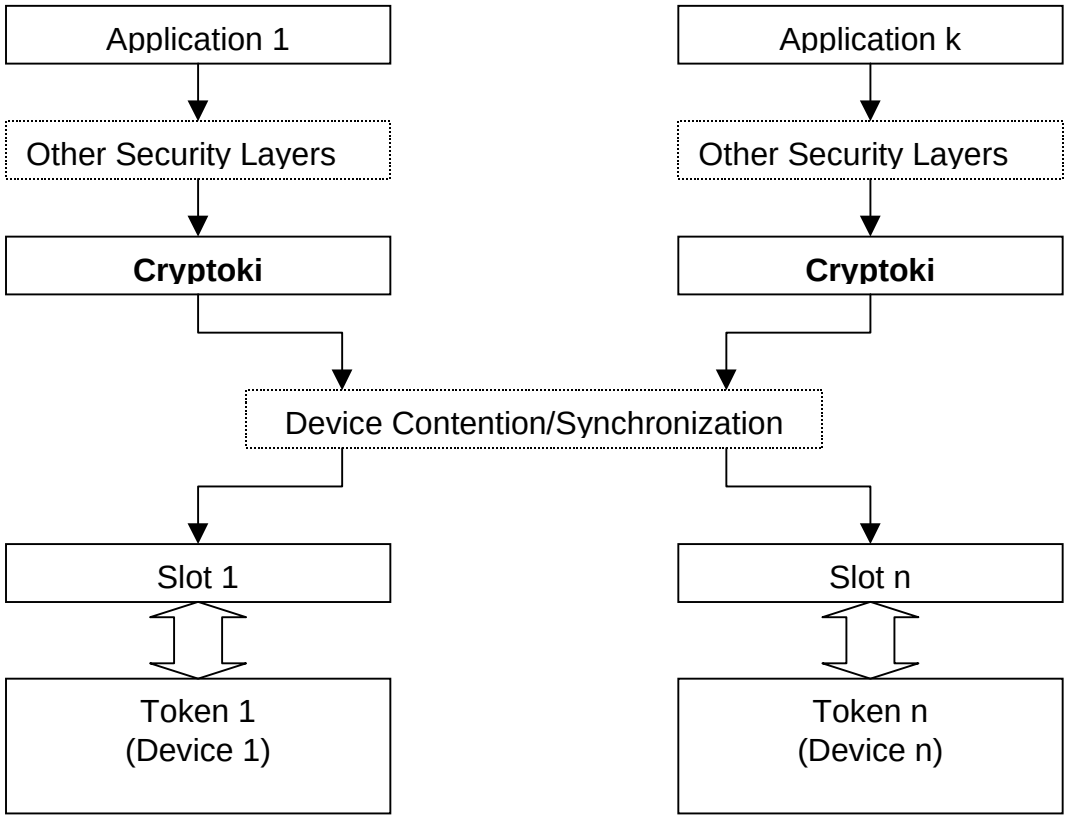


Figura 3 Modelo General PKCS#11

Cryptoki provee una interfaz a uno o más dispositivos criptográficos que están activos para el sistema a través de *slots*. Cada uno de dichos slots, que representa un lector del dispositivo u otro dispositivo, puede contener o no un *token*. Un token está presente en el slot cuando el dispositivo criptográfico está presente en el lector. Esta es una de las interpretaciones posibles que pueden realizarse debido a que Cryptoki provee también una estructura lógica diferente. Es posible que múltiples slots compartan al mismo tiempo el mismo lector de dispositivo. La conclusión puede ser que un sistema tiene un número limitado de slots y una aplicación puede conectarse a cualquiera o a todos los slots simultáneamente.

Un dispositivo criptográfico puede realizar varias operaciones criptográficas por medio de un conjunto de instrucciones que acepta el dispositivo; dichos comandos son enviados al dispositivo por medio de los drivers de los mismos instalados en el sistema a través de USB, bahías PCMCIA o sockets principalmente. Cryptoki hace que cualquier dispositivo criptográfico se parezca a cualquier otro dispositivo desde el punto de vista lógico, independientemente de la tecnología utilizada o, incluso, sin conocer qué tecnología está involucrada o presente. De esta manera el dispositivo publica todas sus funcionalidades a través de software sin que se requiera de ningún hardware específico en la máquina cliente donde reside la aplicación que quiere hacer uso de los servicios contenidos en el dispositivo.

Cryptoki ha sido implementado como una librería que soporta las funciones de la interfaz de la que harán uso las aplicaciones que quieran tener acceso a él. El tipo y las capacidades o tipos de operación que puede realizar un dispositivo dependen de la versión particular y concreta de Cryptoki a la que se refiera. De esta manera, este estándar especifica la interfaz a la librería, pero no las capacidades o acciones que puede realizar cada dispositivo.

2.2.1.3 Perspectiva lógica del token

Desde el punto de vista lógico el token es un dispositivo que almacena objetos que pueden realizar funciones criptográficas. Cryptoki define tres clases de objetos: datos, certificados y claves. Datos son los objetos definidos por la aplicación. Un objeto de certificado almacena un certificado. Un objeto de clave almacena una clave criptográfica o de seguridad. Dicha clave puede ser de cualquier tipo (pública, privada o simétrica). La siguiente figura muestra el token desde la perspectiva lógica (13):

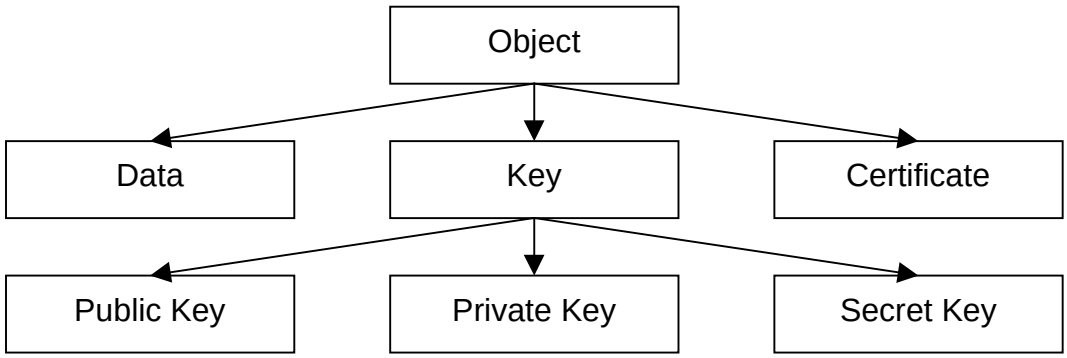


Figura 4 Visión lógica del token

Los objetos también se clasifican según su tiempo de vida y visibilidad. Se tendrán los objetos propios del token que serán visibles para todas las aplicaciones conectadas con suficiente nivel de permisos y permanecen en el token aunque cuando las sesiones (conexiones entre las diferentes aplicaciones y el token) han sido desconectadas. Los objetos de sesión son más temporales y se eliminan del token cuando se produce la desconexión del ordenador. Además estos objetos de sesión son únicamente visibles a la aplicación que los creó. Se puede tener también objetos públicos y privados (los que son únicamente accesibles por un usuario autenticado mediante PIN o contraseña).

Los tokens tienen por tanto la capacidad de crear, destruir, manipular y buscar objetos en el token. Puede, de igual modo, realizar funciones criptográficas y debe tener un generador interno de números aleatorios.

Cobra aquí importancia la diferencia entre la perspectiva general del token y su implementación ya que no todos los dispositivos criptográficos utilizarán este concepto de objetos o no podrán realizar todas las funciones criptográficas posibles. Por ello, todos los dispositivos que sean compatibles con Cryptoki adaptarán su funcionamiento interno a la lógica de programación descrita.

2.2.1.4 Usuarios

La actual versión de Cryptoki reconoce dos tipos de usuario diferentes. Uno de ellos es el Security Officer (SO) y el otro de ellos es un usuario normal. Sólo los usuarios normales son capaces de acceder a los objetos privados del token, pero únicamente una vez que se hayan autenticado mediante su contraseña. Algunos dispositivos requieren también de usuarios autenticados para poder acceder a realizar funciones criptográficas. El papel fundamental del SO es inicializar el token y establecer el PIN de usuario. Sin que el SO haya realizado estas operaciones el usuario normal no puede autenticarse por vez primera dentro del token o dispositivo.

2.2.2 RSA

El sistema criptográfico con clave pública RSA (14) es un algoritmo asimétrico basado en cifrado de bloques, que utiliza una clave pública, la cual se distribuye (en forma autenticada preferentemente), y otra privada, la cual es guardada en secreto por su propietario. Una clave, en este contexto, es un número de gran tamaño.

Cuando se quiere enviar un mensaje, el emisor busca la clave pública de cifrado del receptor o clave pública, cifra su mensaje con esa clave, y una vez que el mensaje cifrado llega al receptor, éste se ocupa de descifrarlo usando su clave oculta o privada.

Los mensajes enviados usando el algoritmo RSA se representan mediante números y el funcionamiento se basa en el producto de dos números primos grandes elegidos al azar para conformar la clave de descifrado. La seguridad de este algoritmo radica en que no hay maneras rápidas conocidas de factorizar un número grande en sus factores primos. La computación cuántica podría proveer una solución a este problema de factorización, pero es algo en proceso de desarrollo (14).

El algoritmo fue descrito en 1977 por Ron Rivest, Adi Shamir y Len Adleman en el MIT; las letras RSA son las primeras letras de sus apellidos. Clifford Cocks, un matemático británico trabajando para la agencia de inteligencia británica GCHQ describió un sistema equivalente en un documento interno en 1973. Debido a la lentitud de la implementación en las computadoras de la época, se lo consideró una curiosidad. Su descubrimiento sin embargo no fue revelado hasta 1997 ya que era confidencial (15).

En general las tecnologías de clave pública son más lentas que los algoritmos de clave privada si bien tienen una mayor robustez por ello suelen ser utilizadas en escenarios donde la velocidad de procesado no sea una pieza clave. Estas tecnologías conforman también la base

de las tecnologías de firma digital junto con las funciones de hash que se estudian posteriormente.

La siguiente figura muestra el proceso de cifrado y descifrado mediante RSA de dos usuarios (16).

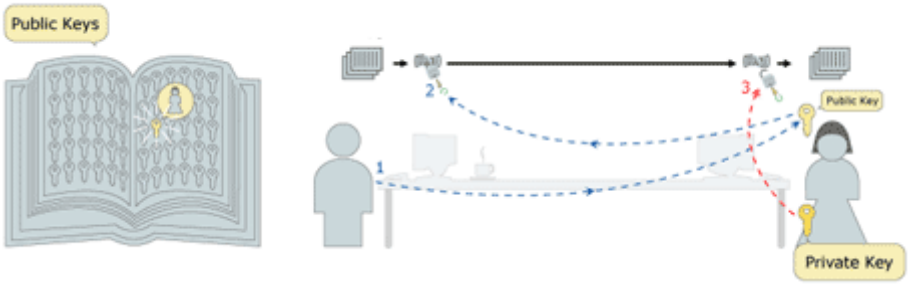


Figura 5 RSA

Matemáticamente RSA tiene el siguiente fundamento matemático (17):

1. El usuario elige dos números primos p y q
2. Se calcula n = p·q
3. Se calcula $\Phi(n) = (p-1)\cdot(q-1)$
4. Se elige e, $1 < e < \Phi(n)$ con la norma de que e y $\Phi(n)$ sean primos relativos
5. Se elige d como el inverso de e módulo $\Phi(n)$
6. Se establece (e, n) como la clave pública del usuario y d como la clave privada de esta manera se tienen los procesos de cifrado y descifrado como:
    a. Cifrado de m: $c = m^e \ mod \ n$
    b. Descifrado de c: $d = m = c^d \ mod \ n$

2.2.3 Rijndael – AES

Advanced Encryption Standard (AES) (18) o estándar de encriptación avanzado, también conocido como Rijndael, es un esquema de cifrado por bloques adoptado como un estándar de cifrado por el gobierno de los Estados Unidos. Se espera que sea usado en el mundo entero y analizado exhaustivamente, como fue el caso de su predecesor, Data Encryption Standard (DES). AES fue anunciado por el Instituto Nacional de Estándares y Tecnología (NIST) como FIPS PUB 197 de los Estados Unidos (FIPS 197) el 26 de noviembre de 2001 después de un proceso de estandarización que duró 5 años. Se transformó en un estándar efectivo el 26 de mayo de 2002. Desde 2006, AES es uno de los algoritmos más populares usados en criptografía simétrica (19).

El cifrador fue desarrollado por dos criptólogos belgas, Joan Daemen y Vincent Rijmen, ambos estudiantes de la Katholieke Universiteit Leuven, y enviado al proceso de selección AES bajo el nombre *Rijndael* (18).

En la actualidad AES consta de tres tamaños de claves diferentes: 128, 192 y 256 bits (20) considerándose 192 y 256 tamaños de clave suficiente para el cifrado de documentos marcados como TOP SECRET.

Estrictamente hablando, AES no es precisamente Rijndael (aunque en la práctica se los llama de manera indistinta) ya que Rijndael permite un mayor rango de tamaño de bloques y longitud de claves. AES tiene un tamaño de bloque fijo de 128 bits y tamaños de llave de 128, 192 ó 256 bits, mientras que Rijndael puede ser especificado por una clave que sea múltiplo de 32 bits, con un mínimo de 128 bits y un máximo de 256 bits. La clave se expande usando el esquema de claves de Rijndael.

La mayoría de los cálculos del algoritmo AES se hacen en un campo finito determinado. AES opera en una matriz de 4×4 bytes, llamada *state* (algunas versiones de Rijndael con un tamaño de bloque mayor tienen columnas adicionales en el state). Para el cifrado, cada ronda de la aplicación del algoritmo AES (excepto la última) consiste en cuatro pasos (19). A continuación se muestra una breve explicación de las operaciones que se realizan en cada uno de los pasos así como el pseudocódigo necesario para la generación de cifrados utilizando AES:

```
Cipher(byte in[4*Nb], byte out[4*Nb], word w[Nb*(Nr+1)])
begin
	byte state[4,Nb]
	state = in
	AddRoundKey(state, w[0, Nb-1]) // See Sec. 5.1.4
	for round = 1 step 1 to Nr–1
		SubBytes(state) // See Sec. 5.1.1
		ShiftRows(state) // See Sec. 5.1.2
		MixColumns(state) // See Sec. 5.1.3
		AddRoundKey(state, w[round*Nb, (round+1)*Nb-1])
	end for
	SubBytes(state)
	ShiftRows(state)
	AddRoundKey(state, w[Nr*Nb, (Nr+1)*Nb-1])
	out = state
end
```

En un primer momento se realizan durante 10, 12 o 14 veces (dependiendo del tamaño de la clave) los pasos que 1, 2, 3 y 4. Finalmente la ronda final elimina la operación de MixColumns.

1. SubBytes — en este paso se realiza una sustitución no lineal donde cada byte es reemplazado con otro de acuerdo a una tabla de búsqueda.
2. ShiftRows — en este paso se realiza una transposición donde cada fila del state es rotada de manera cíclica un número determinado de veces.
3. MixColumns — operación de mezclado que opera en las columnas del state, combinando los cuatro bytes en cada columna usando una transformación lineal.
4. AddRoundKey — cada byte del state es combinado con la clave round; cada clave «round» se deriva de la clave de cifrado usando una iteración de la clave.

Para el proceso de descifrado el proceso es similar y el pseudocódigo para descifrar un cifrado AES puede observarse a continuación (19):

```
InvCipher(byte in[4*Nb], byte out[4*Nb], word w[Nb*(Nr+1)])
begin
```

```
        byte state[4,Nb]
        state = in
        AddRoundKey(state, w[Nr*Nb, (Nr+1)*Nb-1]) // See Sec. 5.1.4
        for round = Nr-1 step -1 downto 1
                InvShiftRows(state) // See Sec. 5.3.1
                InvSubBytes(state) // See Sec. 5.3.2
                AddRoundKey(state, w[round*Nb, (round+1)*Nb-1])
                InvMixColumns(state) // See Sec. 5.3.3
        end for
        InvShiftRows(state)
        InvSubBytes(state)
        AddRoundKey(state, w[0, Nb-1])
        out = state
end
```

2.2.3.1 *Etapa SubBytes - Substitución de bits*

En la etapa SubBytes, cada byte en la matriz es actualizado usando la caja-S de Rijndael de 8 bits. Esta operación provee la no linealidad en el cifrado. La caja-S utilizada proviene de la función inversa alrededor del GF(28), conocido por tener grandes propiedades de no linealidad.

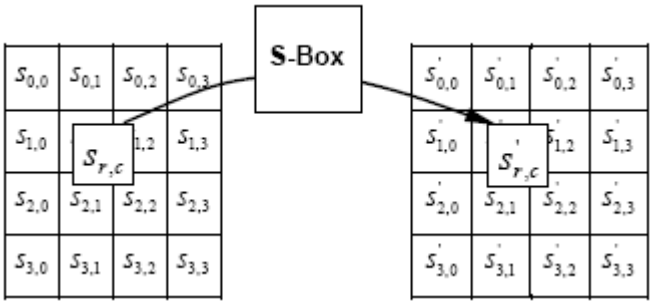


Figura 6 AES: Etapa SubBytes

Para evitar ataques basados en simples propiedades algebraicas, la caja-S se construye por la combinación de la función inversa con una transformación afín invertible. La caja-S también se elige para evitar puntos estables, y también cualesquiera puntos estables opuestos.

		y															
		0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
x	0	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
	1	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
	2	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
	3	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
	4	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
	5	53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
	6	d0	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
	7	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
	8	cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
	9	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
	a	e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
	b	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08
	c	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
	d	70	3e	b5	66	48	03	f6	0e	61	35	57	b9	86	c1	1d	9e
	e	e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
	f	8c	a1	89	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16

Figura 7 AES: Matriz State

2.2.3.2 Etapa ShiftRows -Desplazar filas

El paso ShiftRows opera en las filas del state rotando de manera cíclica los bytes en cada fila por un determinado offset. En AES, la primera fila queda en la misma posición. Cada byte de la segunda fila es rotado una posición a la izquierda. De manera similar, la tercera y cuarta filas son rotadas por los offsets de dos y tres respectivamente. De esta manera, cada columna del State resultante del paso ShiftRows está compuesta por bytes de cada columna del State inicial (variantes de Rijndael con mayor tamaño de bloque tienen offsets distintos).

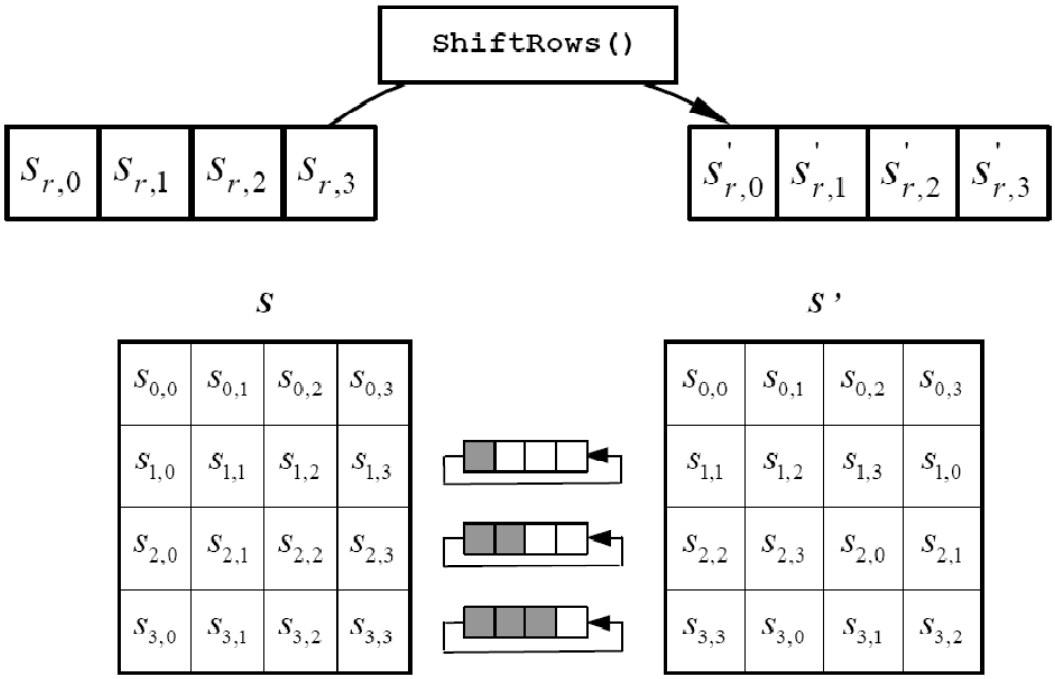


Figura 8 AES: Etapa ShiftRows

2.2.3.3 Etapa MixColumns - Mezclar columnas

En el paso MixColumns, los cuatro bytes de cada columna del state se combinan usando una transformación lineal invertible. La función MixColumns toma cuatro bytes como entrada y devuelve cuatro bytes, donde cada byte de entrada influye todas las salidas de cuatro bytes. Junto con ShiftRows, MixColumns implica difusión en el cifrado. Cada columna se trata como un polinomio GF (28) y luego se multiplica el módulo $x^4 + 1$ con un polinomio fijo c(x). El paso MixColumns puede verse como una multiplicación matricial en el campo finito de Rijndael.

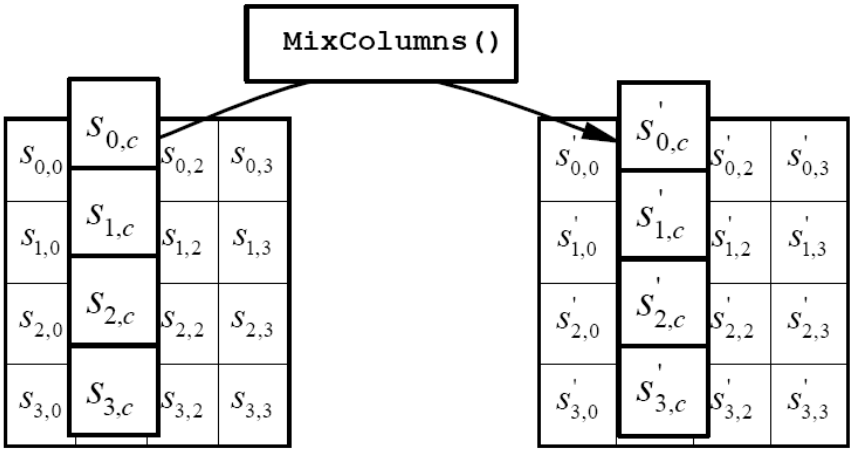


Figura 9 AES: Etapa MixColumns

2.2.3.4 Etapa AddRoundKey- Cálculo de las subclaves

En el paso AddRoundKey, la subclave se combina con el state. En cada ronda se obtiene una subclave de la clave principal, usando la iteración de la clave; cada subclave es del mismo tamaño del state. La subclave se agrega combinando cada byte del state con el correspondiente byte de la subclave usando XOR. Esta operación se realiza tanto en el cifrado como en el descifrado ya que al estar basado en una operación XOR esta etapa es su propia inversa también.

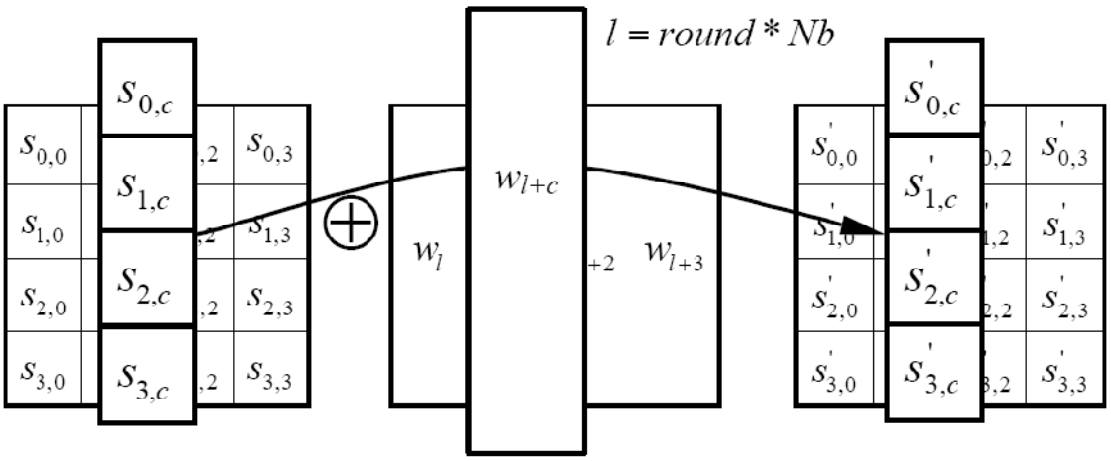


Figura 10 AES: Etapa AddRoundKey

2.2.3.5 *Etapa InvShiftRows*

Es la etapa inversa a *ShiftRows*. Los bytes de las últimas tres filas del *State* se desplazan cíclicamente con diferentes offsets.

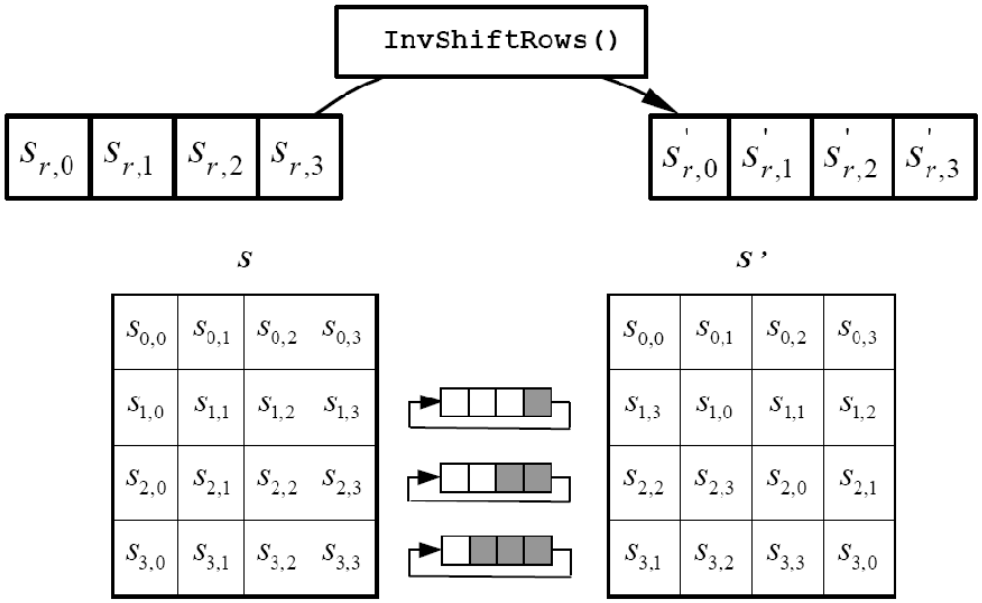


Figura 11 AES: Etapa InShiftRows

2.2.3.6 *Etapa InvSubBytes*

Al igual que la anterior es la etapa inversa a *SubBytes* a la que se la aplica la matriz S inversa de transformación a cada byte del State. La matriz S que se aplica en esta etapa puede observarse en la siguiente figura.

		y															
		0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
x	0	52	09	6a	d5	30	36	a5	38	bf	40	a3	9e	81	f3	d7	fb
	1	7c	e3	39	82	9b	2f	ff	87	34	8e	43	44	c4	de	e9	cb
	2	54	7b	94	32	a6	c2	23	3d	ee	4c	95	0b	42	fa	c3	4e
	3	08	2e	a1	66	28	d9	24	b2	76	5b	a2	49	6d	8b	d1	25
	4	72	f8	f6	64	86	68	98	16	d4	a4	5c	cc	5d	65	b6	92
	5	6c	70	48	50	fd	ed	b9	da	5e	15	46	57	a7	8d	9d	84
	6	90	d8	ab	00	8c	bc	d3	0a	f7	e4	58	05	b8	b3	45	06
	7	d0	2c	1e	8f	ca	3f	0f	02	c1	af	bd	03	01	13	8a	6b
	8	3a	91	11	41	4f	67	dc	ea	97	f2	cf	ce	f0	b4	e6	73
	9	96	ac	74	22	e7	ad	35	85	e2	f9	37	e8	1c	75	df	6e
	a	47	f1	1a	71	1d	29	c5	89	6f	b7	62	0e	aa	18	be	1b
	b	fc	56	3e	4b	c6	d2	79	20	9a	db	c0	fe	78	cd	5a	f4
	c	1f	dd	a8	33	88	07	c7	31	b1	12	10	59	27	80	ec	5f
	d	60	51	7f	a9	19	b5	4a	0d	2d	e5	7a	9f	93	c9	9c	ef
	e	a0	e0	3b	4d	ae	2a	f5	b0	c8	eb	bb	3c	83	53	99	61
	f	17	2b	04	7e	ba	77	d6	26	e1	69	14	63	55	21	0c	7d

Figura 12 AES: Matriz State Descifrado

2.2.3.7 *Etapa InvMixColumns*

Esta etapa es la inversa de *MixColumns*. Opera en la matriz State columna por columna tratando cada una de las columnas como un polinomio de cuarto grado que opera en el GF(2^8), Básicamente esta etapa puede expresarse como una multiplicación de matrices tal y como se observa en la siguiente figura:

$$s'(x) = a^{-1}(x) \otimes s(x) :$$

$$\begin{bmatrix} s'_{0,c} \\ s'_{1,c} \\ s'_{2,c} \\ s'_{3,c} \end{bmatrix} = \begin{bmatrix} 0e & 0b & 0d & 09 \\ 09 & 0e & 0b & 0d \\ 0d & 09 & 0e & 0b \\ 0b & 0d & 09 & 0e \end{bmatrix} \begin{bmatrix} s_{0,c} \\ s_{1,c} \\ s_{2,c} \\ s_{3,c} \end{bmatrix} \quad \text{for } 0 \le c < \boldsymbol{Nb}.$$

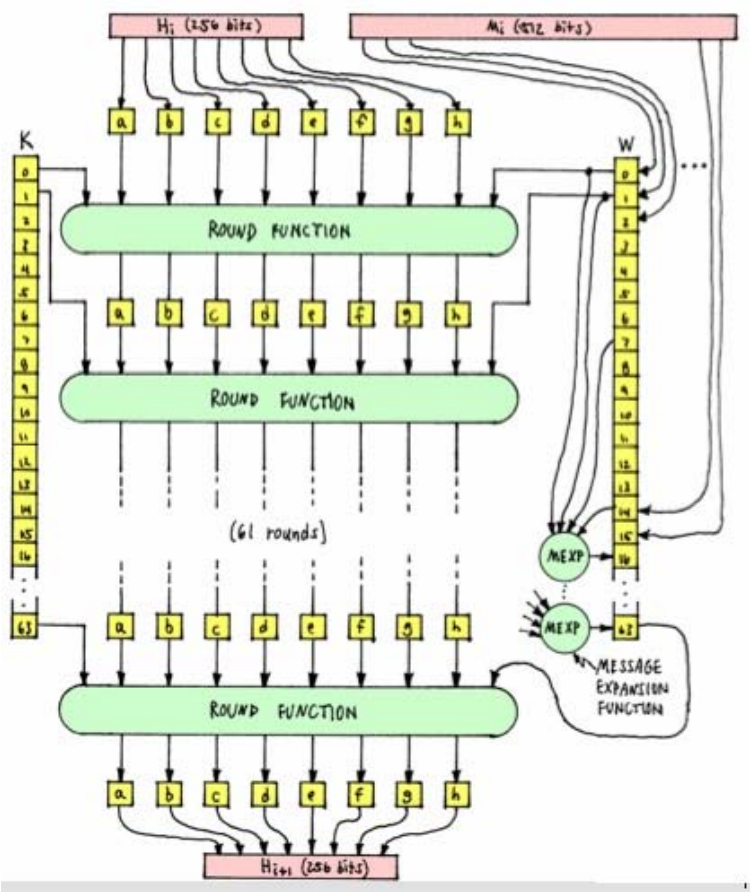
Figura 13 Etapa InvMixColumns

2.2.4 SHA

La familia SHA (Algoritmo de Hash Seguro, de sus siglas en inglés *Secure Hash Algorithm*) (21) es un sistema de funciones hash criptográficas relacionadas de la Agencia de Seguridad Nacional de los Estados Unidos y publicadas por el NIST. El primer algoritmo fue publicado en 1993 y es el conocido oficialmente bajo el nombre de SHA. Sin embargo, hoy día de manera extraoficial, se le llama SHA-0 para evitar confusiones con sus sucesores (21). Dos años más tarde el primer sucesor de SHA fue publicado con el nombre de SHA-1. Existen cuatro variantes más que se han publicado desde entonces cuyas diferencias se basan en un diseño algo modificado y rangos de salida incrementados: SHA-224, SHA-256, SHA-384, y SHA-512 (llamándose SHA-2 a todos ellos).

SHA-1 es el mejor de las funciones hash existentes, y es empleado en varios de los protocolos y aplicaciones más usados. En 2005, se identificaron algunos defectos de seguridad en SHA-1, lo que indicaba que podría haber algún punto débil en la matemática y que era deseable desarrollar una función hash más sólida. Aunque aún no se ha informado de ataques en las variantes de SHA-2 el hecho de ser algorítmicamente similares a SHA-1 hace que se estén dedicando esfuerzos en la mejora de alternativas. Un nuevo estándar, SHA-3 está actualmente en desarrollo.

La siguiente figura muestra un ejemplo práctico de funcionamiento de SHA (22).



2.2.4.1 SHA-0 y SHA-1

La especificación original del algoritmo fue publicada en 1993 como Secure Hash Standard, FIPS PUB 180, por la agencia de estándares NIST (National Institute of Standards and Technology) del gobierno de los Estados Unidos. Esta versión es conocida como SHA-0. Fue retirada por la NSA (National Security Agency) poco después de su publicación y suplantada por una version revisada, publicada en 1995 en FIPS PUB 180-1 y comúnmente conocida como SHA-1. Esta nueva versión difiere con el SHA-0 en una rotación de bits en el mensaje de ubicación en su función de compresión. SHA-1 apareció en el mercado, según la NSA, para corregir un defecto en el algoritmo original que reducía su seguridad criptográfica. De cualquier manera, la NSA no proporcionó ninguna explicación o identificación de que el defecto fuese corregido. Se ha notificado una debilidad tanto en SHA-0 como en SHA-1, aunque este último parece ser más resistente a ataques, reforzando la afirmación de la NSA de que el cambio incrementó la seguridad.

SHA-1 y SHA-0 produce un código cifrado de 160 bits a partir de un mensaje de (264 – 1) bits. SHA-1 está basado en principios similares a los usados por Ronald L. Rivest del MIT en el diseño de los algoritmos de código cifrado MD4 y MD5, pero tiene un diseño más conservador.

2.2.4.2 Familia SHA-2

El NIST publicó cuatro funciones hash adicionales en la familia SHA, cuyo nombre correspondía a la longitud en bits del cifrado: SHA-224, SHA-256, SHA-384, y SHA-512. Los algoritmos son comúnmente conocidos como SHA-2.

Los algoritmos fueron primeramente publicados en 2001 en el borrador FIPS PUB 180-2, al mismo tiempo que la revisión y los comentarios fueron aceptados. FIPS PUB 180-2, que también incluye SHA-1, fue puesto a la venta como estándar oficial en 2002. En febrero de 2004, una nota de cambio fue publicado por FIPS PUB 180-2, especificando una variante, SHA-224, definido para ajustarse a la longitud de clave del Triple DES. Estas variantes están patentadas en Estados Unidos con derechos de licencia gratuita. SHA-256 y SHA-512 son funciones hash nuevas de 32 y 64 palabras, respectivamente. Usan diferente número de desplazamientos y constantes aditivas, pero por lo demás sus estructuras son virtualmente idénticas, difiriendo únicamente en el número de rondas. SHA-224 y SHA-384 simplemente son versiones truncadas de la versión dos, computada con diferentes valores iniciales.

Estas funciones hash no son ampliamente usadas como SHA-1, pero a pesar de eso aparentan proporcionar mucha mejor seguridad. Esto se debe principalmente a que protocolos como SSL no facilitan introducir nuevas funciones hash sin romper el retraso de compatibilidad. Se usa SHA-256 para autentificar los paquetes de software de Debian Linux y en estándares de firmado de mensaje DKIM (*DomainKeys Identified Mail*). SHA-512 es parte de un subsistema para autenticar archivos de video del Tribunal de Crimen Internacional del Genocidio.

2.2.4.3 SHA-3

Se anunció un concurso abierto para una nueva función SHA-3 en el Registro Federal el 2 de noviembre de 2007. NIST está iniciando un esfuerzo por desarrollar uno o más modelos adicionales a los algoritmos hash a través de un concurso público, similar al desarrollo de los Estándares de Encriptación Avanzados (AES, del inglés Advanced Encryption Standard). Las propuestas fueron hasta el 31 de octubre de 2008 y la proclamación del ganador y la publicación del nuevo estándar tendrán lugar en 2012.

2.2.4.4 Aplicaciones

SHA-1 es el algoritmo de la familia SHA más utilizado (23). Forma parte de varias aplicaciones de seguridad y se integra como parte de protocolos ampliamente utilizados como TLS, SSL, PGP, SSH, S/MIME e IPsec. Cualquiera de estos protocolos y aplicaciones también puede utilizar MD5 ya que MD5 y SHA-1 son ambos descendentes de un algoritmo más antiguo denominado MD4.

SHA-1, SHA-224, SHA-256, SHA-384 y SHA-512 son los algoritmos de hash requeridos por ley para su utilización en algunas aplicaciones gubernamentales para la protección de información sensible no clasificada. SHA-1 sin embargo aunque ampliamente utilizada se ha considerado como insegura para aplicaciones federales y se utiliza la familia de protocolos SHA-2.

Una primera motivación para la publicación de SHA fue el desarrollo del DSS (Digital Signature Standard o, del castellano, estándar de firma digital) dentro del cual se ha incorporado.

2.2.4.5 Criptoanálisis y validación

Para una función para la cual L es el número de bits del hash resultante, encontrar un mensaje que se corresponda con el mensaje original puede ser siempre encontrado por fuerza bruta en 2^L intentos. A esto se le conoce con el nombre de ataque imagen y puede ser o no práctico dependiendo del tamaño L y del entorno en el que se esté trabajando. El segundo criterio es el

de colisión, esto es, dos mensajes diferentes que proporcionan el mismo hash de salida, requiere una media de $2^{L/2}$ intentos.

3 Diseño del sistema

El objetivo fundamental es crear una infraestructura en la que los usuarios sean capaces de acceder a todas sus aplicaciones corporativas en los diferentes sistemas operativos que las soportan por medio de un único PIN de acceso al dispositivo seguro.

3.1 Requisitos

El requisito único era crear una arquitectura que permitiese a los usuarios acceder a todas las aplicaciones corporativas de manera transparente, asemejando la solución final a una arquitectura de *Single Sign On* (SSO) con ciertas particularidades. La arquitectura debía ser sencilla, manejable, escalable y fácilmente integrable en el portal de las aplicaciones del cliente.

El segundo requisito era la creación de un escenario de recuperación de todo el contexto de cada usuario en caso de pérdida del dispositivo contenedor de las claves. Este requisito es el que motivó la elección de una arquitectura propietaria en vez de la implantación de soluciones comerciales.

De todas las posibles tecnologías de directorio se utiliza ADAM ya que es la tecnología implantada en cliente y RSA por ser la tecnología de clave asimétrica compatible con .NET. La solución final provee a los usuarios con un par de claves RSA propio con el que se cifran las contraseñas del resto de aplicaciones corporativas a las que cada usuario tiene acceso. Cada una de las contraseñas de cada una de las aplicaciones se guarda en el directorio ADAM y en un fichero de texto en el ordenador local del usuario de manera que sirva de caché en caso de pérdida de conexión con el servidor de directorio. El par de claves RSA, como se ha comentado anteriormente, se guarda en un dispositivo de almacenamiento externo seguro.

Como consecuencia del segundo requisito, en caso de pérdida o hurto de dicho dispositivo, se debe poder realizar una clonación del mismo de manera que no sea necesario recrear un usuario y contraseña en todas las aplicaciones corporativas.

En resumen, se puede concluir que se debe proveer a cada usuario de los siguientes servicios:

- Creación del par de claves RSA.
- Almacenamiento seguro del par de claves en un servidor seguro de manera que permitan su recuperación controlada.
- Acceso fácil y seguro al par de claves RSA creadas en el dispositivo de seguridad.
- Recuperación del par de claves en caso de extravío del dispositivo de almacenamiento seguro.
- Almacenamiento de las claves de aplicación cifradas en el repositorio ADAM.
- Caché de las contraseñas cifradas para que se puedan utilizar en caso de no conexión con el servidor ADAM.
- Protección del dispositivo seguro mediante clave de acceso de usuario o PIN y gestión del mismo en la infraestructura.

La arquitectura de la aplicación satisface todos los requisitos descritos con anterioridad adecuándoles a la infraestructura de red existente, respetando sobre todo la facilidad de

instalación, administración y escalabilidad de manera que el producto sea atractivo para la compañía.

3.2 Visión General de la API

En el siguiente esquema se muestra el diagrama de bloques de la infraestructura desarrollada. Básicamente la infraestructura final consta de 3 librerías[3] de clases, dos aplicaciones y un servicio web.

El proyecto final por tanto no es una aplicación práctica que utilice el framework sino desarrollar un entorno en el que cualquier aplicación con acceso a las librerías .NET puede utilizar este escenario y proporcionar el acceso a las aplicaciones corporativas a través de un único PIN de usuario, siendo el cliente el encargado de diseñar cuándo y bajo qué circunstancias quiere proporcionar este servicio.

Básicamente las aplicaciones PAOServer y PAOWS junto con el servicio web sirven para dar solución al escenario de pérdida del dispositivo seguro sin necesidad de volver a cifrar las claves de las aplicaciones mantenidas en el ADAM. El motivo es el siguiente, el dispositivo seguro almacena el par de claves RSA con el que se cifran los datos de autenticación de todas las aplicaciones corporativas a las que un usuario tiene acceso, en caso de pérdida o robo, si el par de claves RSA no se hubiese almacenado en ningún lugar sería imposible descifrar dichos datos de autenticación y se tendrían que crear nuevas contraseñas para el usuario en todos los sistemas. Del almacenamiento seguro de dichas claves RSA se encargan estas dos aplicaciones y de distribuir las claves de manera segura se encarga el servicio web de manera que en caso de robo o extravío se pueda clonar el dispositivo anterior aunque éste no esté disponible.

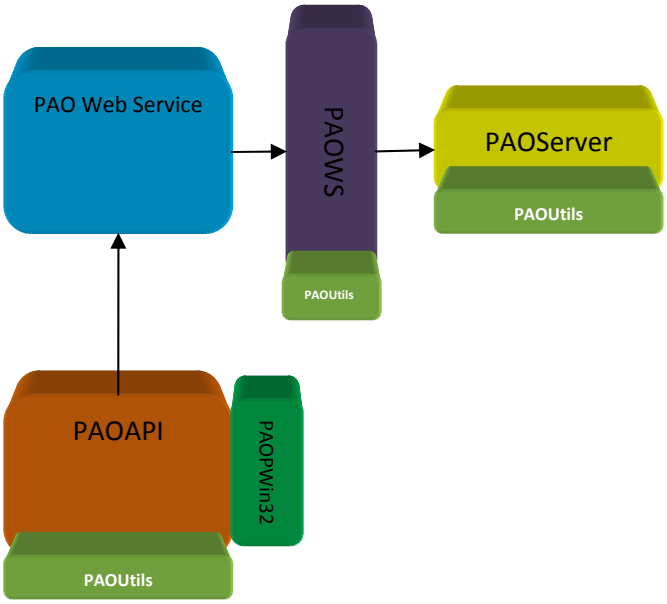


Figura 14 Visión general del API

[3] En este documento librería se utiliza de manera idéntica a como se puede usar la palabra biblioteca y, aunque es erróneo, se ha optado por dicha acepción por ser un término más aceptado entre desarrolladores.

La primera de las librerías es la llamada **PAOUtils.dll**; es la base del resto de librerías y aplicaciones de la API desarrollada. Es consumida por la librería PAOToken y por las aplicaciones PAOServer y PAOWS. Realiza las siguientes funciones:

- Procesado de cifrado y descifrado tanto simétrico como asimétrico.
- Provisión de funciones generales varias para todos los módulos destacando la codificación y decodificación necesaria en las distintas partes durante el proceso de cifrado y descifrado.
- Ejecución de funciones de red. Es decir, una pequeña librería TCP/IP de comunicación con el servidor seguro.
- Depuración precisa y concreta del código para que en caso de error éste sea identificado y subsanado con la mayor brevedad posible.

La segunda de las librerías es **PAOAPI.dll**, es la librería base de las aplicaciones que quieran utilizar los servicios de la infraestructura. Entre las funciones más destacadas se encuentran las siguientes:

- Interfaz al servicio web creado.
- Interfaz a cada una de las operaciones que el PAO permite realizar sobre el dispositivo de almacenamiento de claves de usuario.

La tercera de las librerías es **PAOWin32.dll**, la única desarrollada en C++, su función principal es actuar como interfaz del dispositivo de seguridad que se utilice en la aplicación proporcionando las funciones necesarias para la comunicación a bajo nivel con el mismo.

Las dos aplicaciones desarrolladas están dedicadas a la administración de la arquitectura y no son accesibles por el usuario directamente. La primera de las aplicaciones es **PAOServer.exe**. Es una aplicación servidor dedicada a la administración de la arquitectura cuyas funciones principales son:

- Almacenamiento y mantenimiento seguro de las claves.
- Acceso a las claves de los usuarios por medio de códigos de seguridad temporales.

Por ello a este aplicación se la refiere dentro del documento como servidor de claves o servidor seguro ya que es el centro fundamental de la seguridad dentro del framework y el punto de referencia, en caso de que el usuario pierda de la manera que sea el dispositivo de almacenamiento seguro, para poder realizar una clonación del dispositivo extraviado de manera que el usuario pueda seguir accediendo a sus aplicaciones corporativa.

Para la comunicación con dicha aplicación servidor se crea una aplicación cliente denominada **PAOWS.exe** que envía los comandos a la aplicación del servidor y analiza y presenta la respuesta al mismo.

El servicio web **pao.asmx** expone las funciones de recuperación y almacenamiento seguro a los usuarios de la arquitectura de manera controlada, interoperable y escalable a través de la

librería PAOToken.dll si bien se pueden desarrollar versiones en otros lenguajes que sirvan como interfaz al servicio web.

3.3 La librería PAOAPI.dll

3.3.1 Descripción

El dispositivo de almacenamiento de claves se presenta dentro de la aplicación como algo más complejo que un *simple* dispositivo de almacenamiento seguro. Desde el punto de vista de la lógica de la aplicación debe considerarse más bien como la unión funcional de un dispositivo de almacenamiento y una lógica de la aplicación que permite el desarrollo de las funcionalidades de criptografía asimétrica dentro del contexto de la arquitectura en desarrollo.

Aunque sería posible cifrar cualquier mensaje a partir del objeto PAOToken (objeto principal de toda la librería y clave de la arquitectura ya que es el encargado de actuar como interfaz con el usuario) esta funcionalidad ha sido limitada al ámbito de los procesos listados en el punto anterior. Esto es, si extensiones futuras de la aplicación determinaran que deben cifrarse mensajes en la red o procesos de firma o similares, deberá utilizarse la librería PAOUtils.dll que se describirán más adelante en este documento.

3.3.2 Funciones

La librería PAOToken.dll puede realizar las siguientes funciones sobre el dispositivo USB de Aladdin por medio de la librería PAOWin32.dll:

- Creación e inicialización del dispositivo de usuario
- Cifrado y descifrado de manera asimétrica
- Ejecución del proceso de autenticación
- Cambio del PIN de acceso al dispositivo
- Determinación del número de dispositivos conectados

Es importante remarcar que la aplicación ha sido probada para el dispositivo de Aladdin, sin embargo el acceso a dicho dispositivo se realiza a través de funciones estándar provistas por .NET por lo que cualquier otro dispositivo que soporte la interfaz debería poder adecuarse a la plataforma sin problemas mayores.

Adicionalmente la aplicación puede realizar las siguientes funciones utilizando para ello el dispositivo de usuario cuando los procesos lo requieran:

- Cifrado y descifrado de manera simétrica
- Creación de claves personales
- Recuperación del dispositivo
- Caché de aplicaciones

3.3.3 Creación e inicialización del dispositivo de usuario

El proceso de creación de un dispositivo debe realizarse por el Centro de Atención al Usuario, CAU en adelante, o por el grupo de administración de la seguridad en el dominio. El proceso implica la creación de un contenedor de claves para el par de claves asimétricas generadas, una fecha de creación del dispositivo, un PIN de usuario y una clave simétrica.

La clave privada se cifra con la clave simétrica y se envía al servidor ADAM que se encarga de su almacenamiento. El PIN del usuario se cifra con la clave pública y se envía al servidor ADAM. La clave simétrica se envía mediante canal SSL al servicio web que se encarga de almacenarla en el servidor de claves y gestionar su acceso según el proceso que se describe posteriormente en este documento. La fecha de creación del dispositivo se almacena como parte del nombre del contenedor de claves. Cualquier fallo en el proceso hace que el objeto PAOToken no se cree correctamente y, por tanto, el dispositivo físico no almacenará las claves generadas. Una excepción se producirá en este caso indicándole a la aplicación el motivo concreto de fallo.

Como nota general añadir que los desarrolladores que utilicen la librería deberían capturar las excepciones y gestionar el envío de mensajes personalizados al usuario de la aplicación CAU de administración. Bajo la gestión de estas excepciones el desarrollador puede, opcionalmente, establecer políticas de actuación frente errores como el envío de mensajes a lista de correo de administradores o cualquier otra acción que estime oportuna.

El proceso general de la creación de un dispositivo puede observarse en el siguiente gráfico:

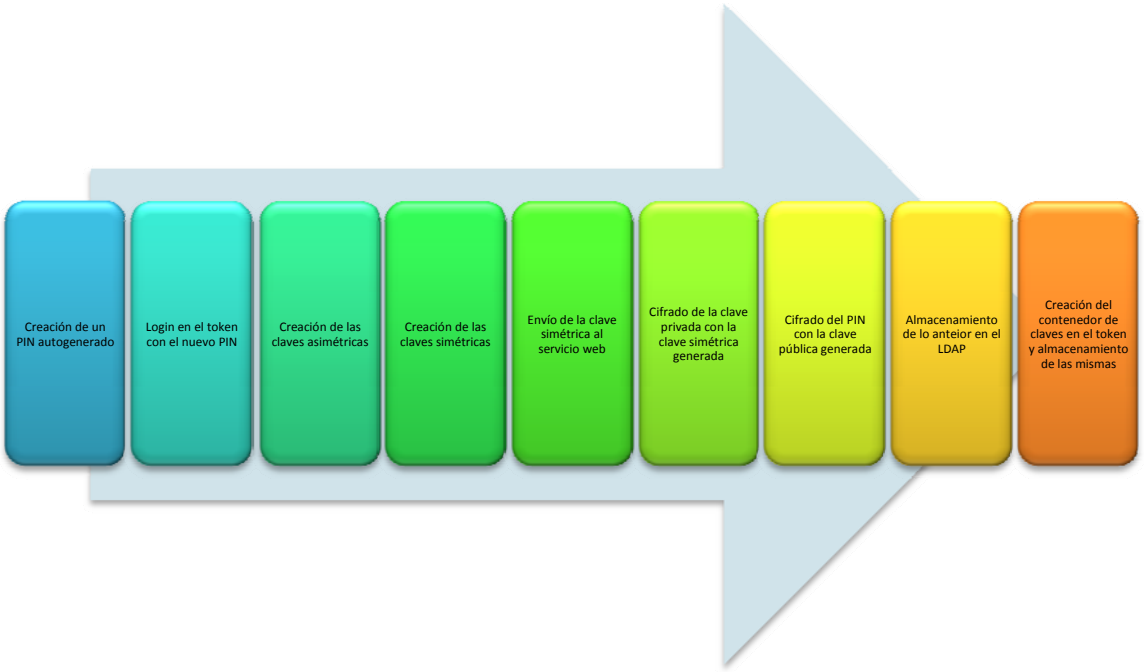


Figura 15 Creación del dispositivo

3.3.4 Cifrado y descifrado de manera asimétrica

Como se ha comentado en la introducción el *PAOToken* puede realizar funciones de cifrado y descifrado de manera asimétrica, si bien esta funcionalidad se ha limitado al ámbito de la aplicación. Esto quiere decir que la aplicación soporta, por ejemplo, el cifrado y descifrado de contraseñas y usuarios de las distintas aplicaciones del entorno de producción de la compañía.

Para el cifrado de manera de asimétrica de mensajes personalizados o cualquier otro tipo de datos adicionales deberá utilizarse la librería *PAOUtils.dll* que se describirá más adelante en este documento. Esto se ha decido para evitar el uso malintencionado o inapropiado del

objeto *PAOToken*, que representa el dispositivo USB de Aladdin con las mencionadas funciones adicionales.

3.3.5 Realizar el proceso de autenticación

La API es capaz de realizar el proceso de autenticación en el dispositivo, mediante la función *OpenToken*. Con la invocación de esta función el desarrollador de la aplicación final autenticará al usuario en el dispositivo de manera que éste pueda acceder a la clave cifrada almacenada en el dispositivo USB, para realizar los procesos de descifrado que se requieran durante la ejecución de la aplicación.

Para una mayor seguridad el PIN se genera automáticamente siguiendo unas políticas de seguridad y una longitud de clave ambas configurables en la API. Este valor es devuelto tanto en texto en claro como cifrado para que se almacene en el ADAM y permita la recuperación del mismo como parte del proceso de recuperación del dispositivo. La librería acepta especificar el PIN en el momento de la creación del dispositivo en el caso de que se trate de una clonación debido a pérdida o hurto.

Remarcar que el PIN del usuario únicamente es necesario para las labores de descifrado y creación y eliminación del contenedor de claves. Como parte del proceso de autenticación se comprueba que el dispositivo tenga un contenedor válido creado y, en caso afirmativo, que el dispositivo sea el último que haya sido creado para el usuario por el CAU y que el usuario autenticado en Windows sea el mismo que el usuario que posee el dispositivo. Las utilidades necesarias para la comprobación de dichas especificaciones se desarrollan en la librería *PAOUtils.dll*. El proceso de autenticación y validación de la información es como se muestra a continuación:

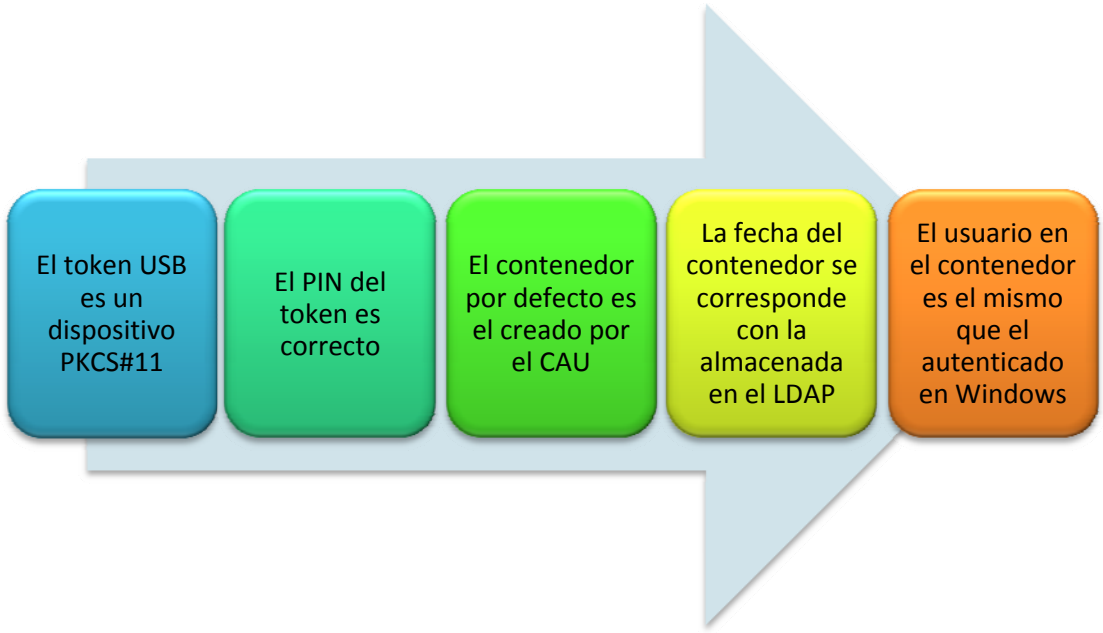


Figura 16 Proceso de autenticación y validación

Si todo el proceso es correcto se entiende que el usuario en uso de la librería *PAOToken.dll* es válido para acceder a cualquiera de las funcionalidades de la arquitectura, que al estar libre de perfiles de usuarios, las funcionalidades expuestas son las mismas para todos.

3.3.6 Cambiar el PIN de acceso al dispositivo

El cambio del PIN del usuario es un proceso que requiere de cierta sincronización con el servidor LDAP debido a las especificaciones del proyecto. Un usuario proveerá tanto su PIN antiguo como su PIN modificado. Como resultado del proceso el usuario obtendrá, en caso satisfactorio, el nuevo PIN cifrado con la clave pública del usuario y, evidentemente, se realizará dicho cambio de PIN en el dispositivo de usuario.

3.3.7 Cifrar y descifrar de manera simétrica

El proceso de cifrado y descifrado simétrico se realiza mediante lógica adicional de la aplicación. La clave simétrica generada durante el proceso de creación del token es utilizada para el almacenamiento de manera segura del par de claves asimétricas generadas. El cifrado simétrico utilizado es AES-256, con modo de operación CBC y relleno con ceros. Al igual que la clave, el vector de inicialización se genera de manera aleatoria y se almacena de manera segura en el servidor de claves. Por tanto, al igual que en el caso asimétrico, el cifrado y descifrado de mensajes o datos adicionales debe realizarse por medio de la librería *PAOUtils.dll* que, como se verá, expone dicha funcionalidad.

3.3.8 Creación de claves personales

Como se ha comentado anteriormente, parte del proceso de creación del dispositivo genera un vector de inicialización y una clave aleatoria que se utilizarán como parte del cifrado AES necesario para el almacenamiento seguro de las claves dentro del servidor de directorio. Además cada usuario se conecta a la aplicación PAO y al dispositivo USB de Aladdin utilizando un PIN aleatorio generado por la aplicación.

El usuario posee adicionalmente un par de claves pública y privada asimétricas. Nótese que este par de claves no son un certificado digital, ya que no está firmado por ninguna autoridad de certificación, en adelante CA. El cómo se publiquen las claves públicas queda bajo la responsabilidad final de la aplicación que utilice esta API, si bien, se proveen todos los mecanismos necesarios para el uso seguro de las mismas dentro de la librería desarrollada.

Por resumir, el usuario dentro de la librería debería poseer tres tipos de claves: la simétrica, la asimétrica y el PIN. El usuario únicamente debe recordar su PIN de acceso al dispositivo que le permitirá acceder a las funcionalidades criptográficas descritas.

3.3.9 Recuperar el dispositivo

La librería permite la clonación de un dispositivo en caso de pérdida, hurto u olvido. El primer paso es obtener la clave simétrica que se asignó al usuario. Para ello la librería se conecta al servicio web mediante SSL y obtiene la clave en texto plano. Es muy importante que el servicio web se exponga a través de servidor web seguro, en caso contrario, la arquitectura expone una brecha en la seguridad crítica. Una vez obtenida la clave simétrica se conecta al servidor ADAM donde recupera el par de claves pública y privada cifradas simétricamente y el PIN del usuario cifrado asimétricamente.

La librería descifra la información del par de claves y las introduce en el nuevo token de manera que usuario podrá continuar accediendo a toda la información almacenada en el servidor LDAP o en su fichero de caché almacenado en su ordenador.

Únicamente se puede acceder al proceso de recuperación del usuario autenticado en el dominio en la sesión actual, evitando de esta manera que un usuario pueda clonar el dispositivo de otro. Además como se comentó durante el proceso de creación se consideraba como parte del contenedor de claves la fecha en la que el dispositivo fue creado. Esto es así, para que en el caso de pérdida otro usuario o el mismo no pueda acceder por error con un dispositivo o bien que no sea el suyo o bien que no es el dispositivo más reciente que tiene asignado, forzándole en cierto modo a que devuelva uno de los dispositivos.

Por tanto, en el momento de creación de un nuevo dispositivo el anterior queda inhabilitado. Notése que en caso de que el CAU quiera denegar el acceso de un dispositivo o un usuario, lo único que debe hacer es modificar la fecha de creación del dispositivo por una diferente en el servidor ADAM. De esta manera, el proceso de validación del dispositivo fallará y el usuario no podrá acceder a la información almacenada a través de la librería. Obviamente, desde el punto de vista de seguridad de directorios el negar el acceso del usuario del dominio al servidor de directorio tiene idéntica repercusión.

El proceso de recuperación del dispositivo se muestra en la siguiente figura.

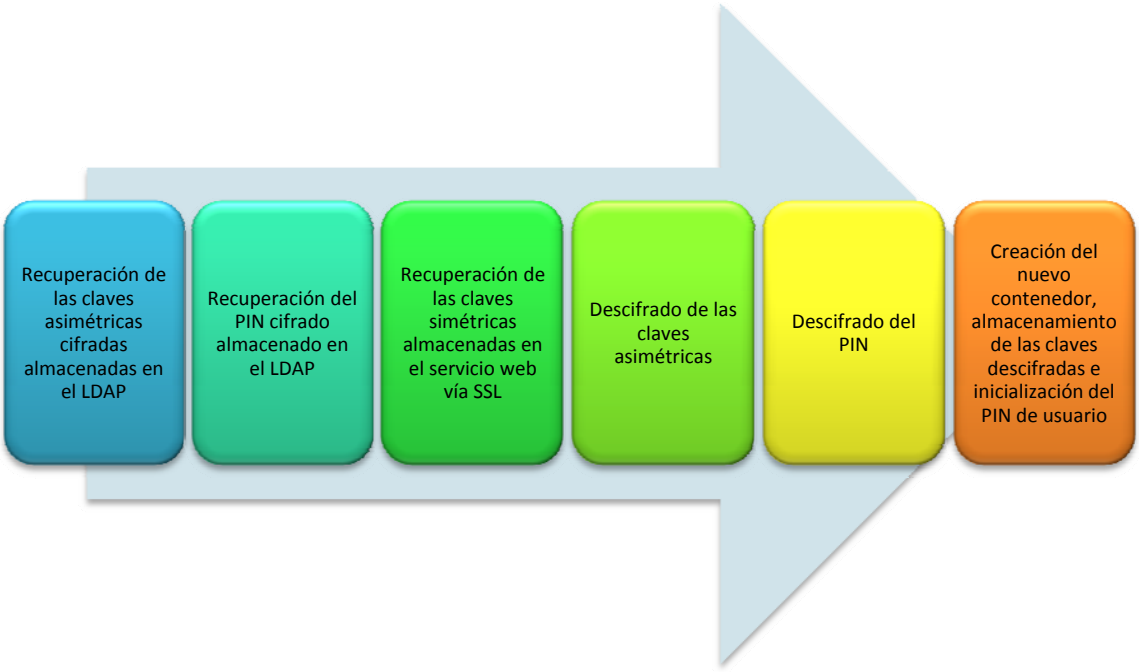


Figura 17 Recuperación del dispositivo

3.3.10 Caché de aplicaciones

La librería, a través del objeto *PAOToken*, es capaz de almacenar, modificar y obtener información almacenada en un fichero local. Estas funcionalidades se consiguen mediante las invocaciones de las funciones *CreateCacheFile*, *AddAppToCache* y *GetLoginInfo*. El nombre de cada una de las funciones es auto explicativo en inglés que es el idioma en el que se desarrolló la API a petición de la empresa cliente. En cualquier caso, mediante la primera de las funciones se crea el fichero local con el formato y en la ruta especificada. Mediante la segunda se añade

un par usuario contraseña para una aplicación determinada al ordenador local del usuario y mediante la tercera se recupera el par para una aplicación determinada.

La función *CreateCacheFile*, permite especificar la ruta y el nombre del fichero de cache y, adicionalmente, si se desea sobrescribir en caso de que este fichero ya exista en la ruta y con el nombre especificados. La invocación sin parámetros, la considerada por defecto, crea el fichero de caché en la ruta *%AppData%\PAO\pao.cache.txt* y sobrescribe el fichero en caso de que ya exista. Esta funcionalidad, puede cambiarse a cualquiera de sus variantes por medio de las sobrecargas del método anterior.

La función *AddAppToCache* añade un par de usuario contraseña al fichero de caché. La funcionalidad por defecto de este método no puede cambiarse ya que se ha desarrollado de la manera más general posible ajustándose a los requerimientos y, evitando así, un mal uso de la misma. Las sobrecargas de dicha función únicamente permiten almacenar la información en un fichero de caché diferente al que se encuentre en la ruta por defecto.

La funcionalidad puede observarse en el siguiente diagrama de flujo.

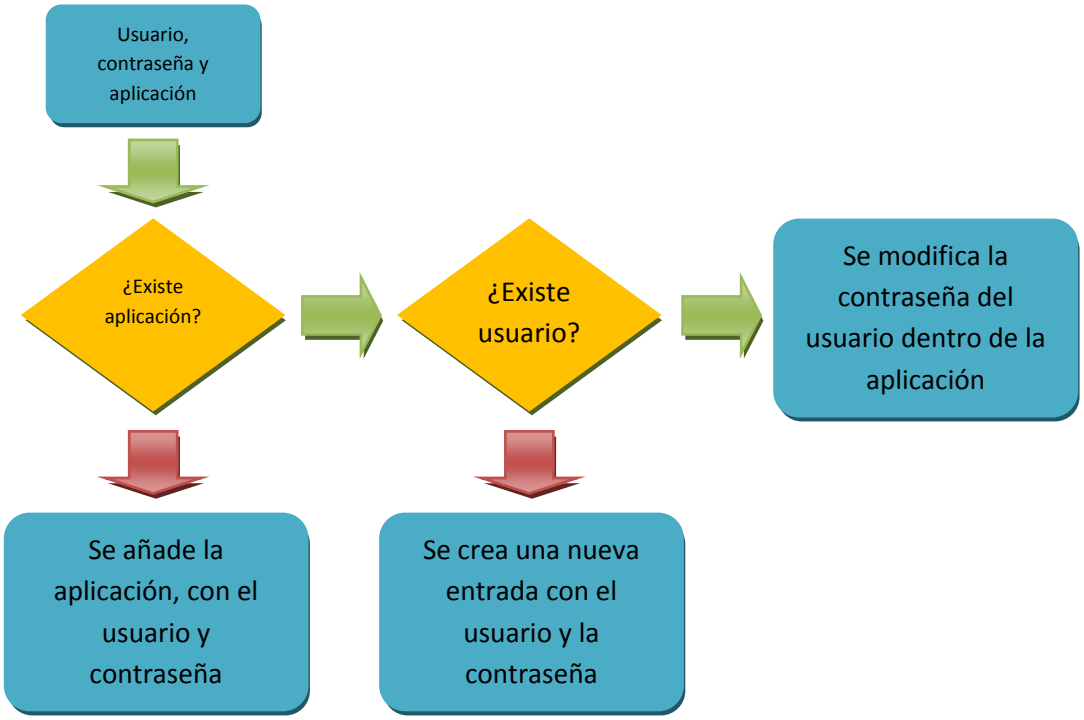


Figura 18 Funcionalidad *AddAppToCache*

La función *GetLoginInfo* devuelve toda la información asociada a una aplicación que exista en el fichero de caché de aplicaciones. En el caso de que el fichero no exista o que la aplicación no exista se devolverá *null*. En caso contrario se devolverá una tabla hash en la que la clave será el nombre de usuario para esa aplicación y el valor asociado con esa clave será la contraseña del usuario para esa aplicación. Remarcar que la información almacenada en esa tabla está en texto plano, por lo que la tabla debería ser eliminada en el mismo momento en el que se

termine de usar los datos que contiene. Se recomienda el uso de la sentencia *using* en caso de que se esté desarrollando la aplicación en C#.

3.3.11 Implementación

En lo sucesivo en el apartado *Descripción del código* se muestran las clases que pertenecen a cada una de las librerías o aplicaciones. Asimismo cada elemento o figura expuesto en estos apartados se ha generado utilizando la opción de ingeniería inversa del producto Microsoft Office Visio 2003. El código entregado en formato electrónico está debidamente comentado y puede generarse en formato MSDN utilizando varias herramientas del mercado de libre distribución, como por ejemplo *Sandcastle*.

De cada uno de los bloques se destacarán brevemente las funciones principales. Por último se presentará un diagrama UML general que detalle la relación entre los elementos.

La clase *PAOToken* es la principal de la librería. A través de ella se cargan los conectores de acceso al dispositivo de almacenamiento seguro y a través de ella se expone toda la funcionalidad de la API o arquitectura propuesta. Entre las diferentes funciones cabe destacar las siguientes:

- `CreatePAOToken`, crea el par de claves RSA dentro del contenedor de claves, la clave simétrica que cifra el par de claves generado e inicializar el PIN de usuario. La clave simétrica es guardada automáticamente en el servidor en caso que la librería haya sido compilada con el símbolo de compilación WEB_SERVICE_INTEGRATION.
- `InitUserPIN`, autentica como usuario SO dentro del dispositivo borrando todo el contenido anterior utilizando la contraseña por defecto y establece la nueva contraseña de acceso al dispositivo generada automáticamente.
- `ChangeTokenPIN`, cambia la contraseña de acceso si la nueva cumple los requerimientos mínimos explicados con anterioridad.
- `OpenToken`, autentica al usuario dentro del dispositivo para que éste no pida la contraseña de forma explícita al usuario.
- `CloseToken`, libera los recursos asignados durante las operaciones de usuario.
- `RecoverPAOToken`, clona un dispositivo a partir de los datos contenidos en el LDAP. Se tiene que poder conectar al servicio web para recuperar la clave simétrica.
- `AsymmetricEncrypt/AsymmetricDecrypt`, cifra cadenas de caracteres a partir de las claves almacenadas en el dispositivo.
- `CreateCacheFile`, crea el fichero de cache en la máquina local.
- `AddAppToCache`, agrega un par usuario-contraseña de cualquier aplicación corporativa a la cache.
- `GetLoginInfo`, recupera todos los pares usuario-contraseña asociados a una aplicación corporativa.
- `GetNumberOfTokens`, recupera el número de dispositivos conectados al ordenador en cualquier momento.

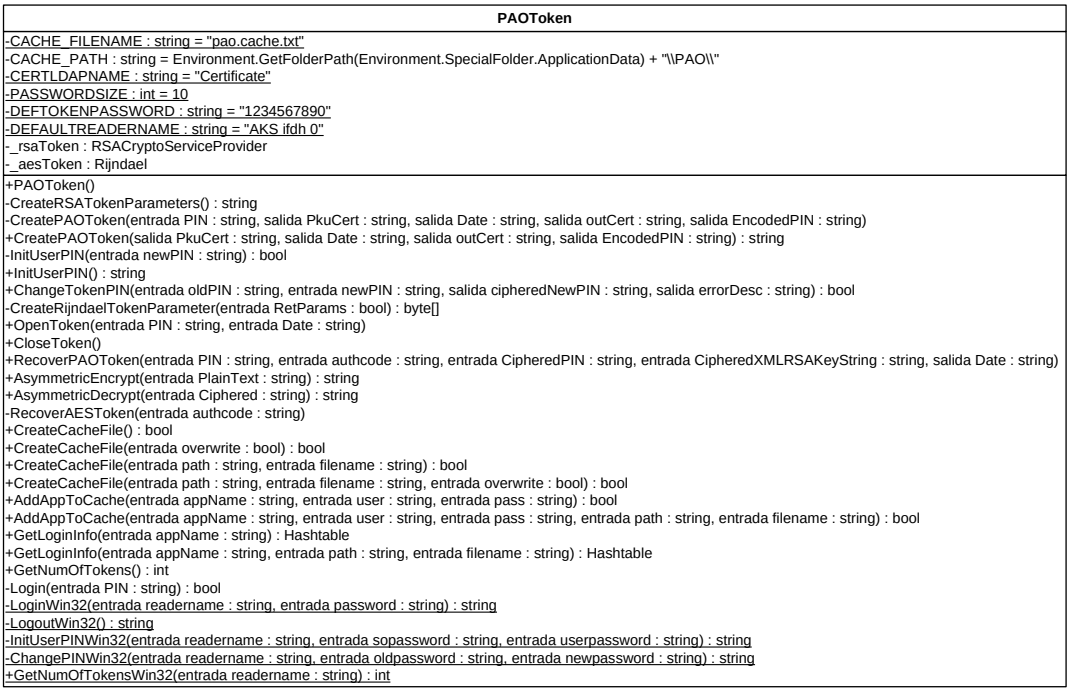


Figura 19 Diagrama UML de PAOToken

La clase *PAOTokenUtils* es una clase estática que contiene la mayoría de las funciones de apoyo que la clase anterior necesita para su correcto funcionamiento. La decisión de implementarla como clase estática se debe básicamente a la no necesidad de crear objetos de la clase para su uso, esto evita que la clase *PAOToken* tenga que tener una propiedad del tipo de dato *PAOTokenUtils*. Entre las funciones a destacar caben las siguientes:

- `LastPAOError`, contiene una descripción detallada del último error que se ha producido en el entorno.
- `CheckTokenOwner`, comprueba que el usuario que creó el dispositivo, en lo que a la arquitectura se refiere y el usuario que está autenticado en la máquina local es el mismo. Comprueba adicionalmente que la fecha de creación que se encuentra almacenada en el dispositivo es la misma que la fecha de creación almacenada en el servidor LDAP, esto se utiliza para evitar que después de una clonación el dispositivo antiguo tenga acceso a los servicios de la plataforma.

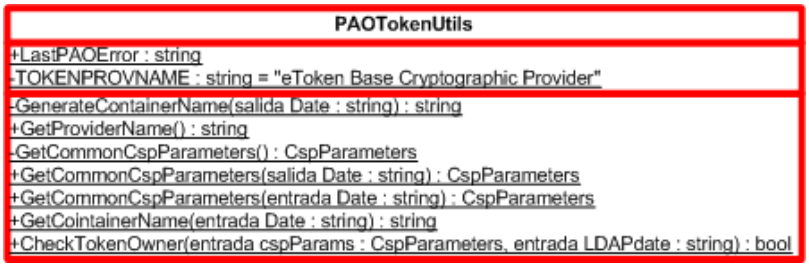


Figura 20 Diagrama UML de PAOTokenUtils

La clase *PAOTokenWS* es una interfaz entre el objeto PAOToken y los servicios web publicados por la arquitectura. Básicamente expone a los objetos de la plataforma las funcionalidades que le permiten hacer una copia de respaldo y recuperar una contraseña simétrica almacenada en el servidor. Por tanto, sus funciones a destacar son únicamente dos:

- `BackupPassword`, realiza una copia de respaldo en el servicio web.
- `GetPassword`, obtiene la clave simétrica almacenada en el servidor.

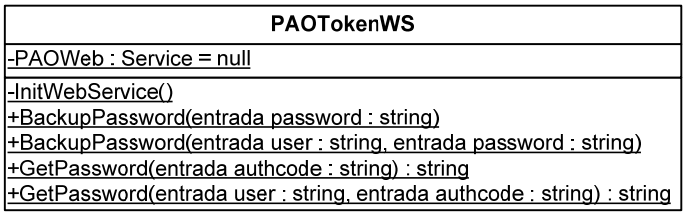


Figura 21 Diagrama UML de PAOTokenWS

A continuación se muestra un diagrama UML por bloques. Como puede observarse la estructura de la librería es muy sencilla. Las interacciones entre las librerías y las aplicaciones se muestran en la figura del apartado *Visión General de la API (3.2)*.

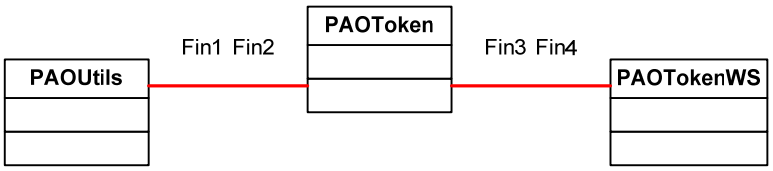


Figura 22 Diagrama UML por bloques

3.4 La librería PAOUtils.dll

3.4.1 Descripción y funciones

La librería *PAOUtils* contiene la base criptográfica sobre la que se apoyan el resto de librerías y aplicaciones del PAO. Básicamente encapsula las llamadas al sistema criptográfico de Windows subyacente o al dispositivo de Aladdin, proporcionando una interfaz clara y sencilla que permita tanto el cifrado y descifrado de manera simétrica y asimétrica como operaciones de hash.

La librería utiliza como método de cifrado simétrico AES-256 y como método de cifrado asimétrico RSA. No es objeto de la librería forzar a utilizar un tamaño de claves o relleno específico por lo que la librería actúa sobre los CSP creados en la aplicación que utiliza la librería. Los CSP, del inglés *CryptoServiceProvider*, contienen toda la información sobre los detalles de la implementación que se va a realizar de una tecnología de seguridad dentro del framework .NET. En general en estos objetos podemos encontrar desde el nombre del contenedor de claves hasta el tamaño de la misma o el relleno que se va a utilizar durante el

cifrado de datos. El caso del hash es contrario a esta política y utiliza SHA-256 como algoritmo de hash dentro de la librería.

Adicionalmente a la librería criptográfica, se desarrolla una librería de utilidades compartidas entre las diferentes aplicaciones. Estas utilidades se dividen en cuatro: las dedicadas a la gestión de contraseñas, las dedicadas a la gestión de datos cifrados, las dedicadas a las operaciones de red y las dedicadas a las operaciones de depuración de la aplicación.

Las dedicadas a la gestión de contraseñas se basan en la creación de ciertas contraseñas automáticas que cumplan unos requisitos mínimos. Los requisitos por defecto para considerar una contraseña o un PIN de dispositivo de usuario como seguro son los que se muestran a continuación:

- Longitud mínima de 10 caracteres
- Utilización de al menos un símbolo no alfanumérico como @, #, ., &, etc.
- Existencia de al menos una letra minúscula, otra mayúscula y un número.

Para modificar el comportamiento por defecto se sugiere modificar la función privada *AddPasswordPolicy* al antojo del desarrollador final. Si bien, las contraseñas generadas por defecto son seguras y no debería suavizarse las políticas de manera que las contraseñas sean menos seguras. Esto se debe a que las contraseñas se usan como claves temporales durante la recuperación del dispositivo de usuario y para la creación de PINs de usuarios de manera automática. Para utilizar esta funcionalidad en la aplicación final debe ejecutarse la función *GenerateUserPIN* especificando, en caso de requerir uno de longitud diferente a la de por defecto, el tamaño de la contraseña que quiere generarse. Para reducir la complejidad del proyecto y con ello el coste se proponen cambios directos sobre el código fuente ya que se eliminó de las especificaciones finales un sistema que permita la carga mediante reflexión la carga y ejecución de librerías externas.

Las funciones dedicadas a la gestión de datos cifrados ayudan a la conversión entre los formatos *string* o cadena de caracteres conteniendo un texto de valores hexadecimales a matriz de bytes y viceversa; en cualquiera de los casos se especifica cuándo se debe y cuándo no eliminar el relleno añadido a los datos cifrados. Estas funciones se han desarrollado fundamentalmente para guardar la información cifrada en formato de cadena de caracteres tanto en el servidor ADAM como en el servidor de claves o en el fichero de caché de contraseñas. Esta funcionalidad se publica al resto de aplicaciones mediantes los métodos *GetStringEncoding* y *GetByteEncoding* especificando la codificación del string que se requiere en bytes o bien si se debe o no eliminar el relleno de los datos cifrados en la matriz de bytes.

Entre las dedicadas a las operaciones de red se encuentra las funcionalidades típicas de cualquier librería de red que son, básicamente, funciones para el envío y la recepción de cadenas de caracteres en un entorno TCP/IP así como la conexión de los clientes al servidor TCP/IP que se le especifique.

La depuración de la aplicación se realiza mediante extensión de los métodos provistos por Microsoft en el paquete *Diagnostics*, que a través de la herencia permite especificar el formato personalizado del texto de salida. En este caso se ha optado por una depuración directa a

fichero de texto en caso de las aplicaciones y las librerías y utilizar los elementos de trazo provistos en la arquitectura ASP.NET en la que se ha desarrollado el servicio web de la aplicación.

3.4.2 Implementación

La librería *PAOUtils.dll* es una librería de apoyo al resto de librerías y es utilizada por la gran mayoría de aplicaciones y librerías dentro de la arquitectura. Por ello posee las funcionalidades de depuración, conexión de red, cifrado y otras funciones de apoyo utilizadas distribuidas en cuatro clases diferentes. Todas las clases excepto la clase encargada de la depuración ofrecen sus funcionalidades a partir de métodos estáticos que no requiere de la creación de los objetos para hacer uso de dichas funcionalidades. El caso de la clase de depuración es un tanto diferente ya que utiliza el mecanismo estándar de depuración de Microsoft .NET Framework que requiere de clases no estáticas.

La clase *PAOCryptoUtil* expone las capacidades de cifrado de la arquitectura. Aunque el dispositivo de almacenamiento seguro únicamente posee la funcionalidad de cifrado asimétrico, la arquitectura también debe ser capaz de cifrar simétricamente y generar hash, por ello esta clase expone todas estas funcionalidades:

- `SymmetricEncrpyt/SymmetricDecrypt`, realiza el cifrado y descifrado simétrico mediante AES.
- `AsymmetricEncrypt/AsymmetricDecrypt`, realiza el cifrado y descifrado asimétrico mediante RSA.
- `GenerateSHA`, genera el hash mediante SHA.

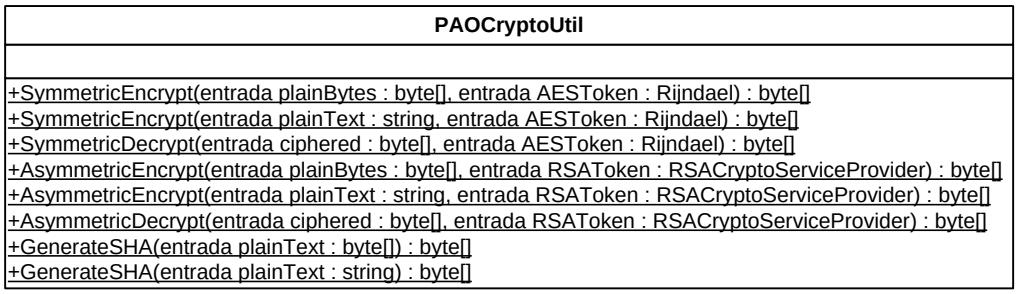


Figura 23 Diagrama UML de PAOCryptoUtil

La clase *PAONet* se encarga de las funciones de red dentro de la arquitectura, que se limitan a establecer una conexión TCP con un servidor y mandar y recibir datos del servidor. Las principales funciones de la librería son:

- `ConnectToServer`, se conecta a un servidor.
- `SendData`, envía datos al servidor a partir de una conexión TCP establecida.
- `ReceiveData`, recibe datos del servidor a partir de una conexión TCP establecida.

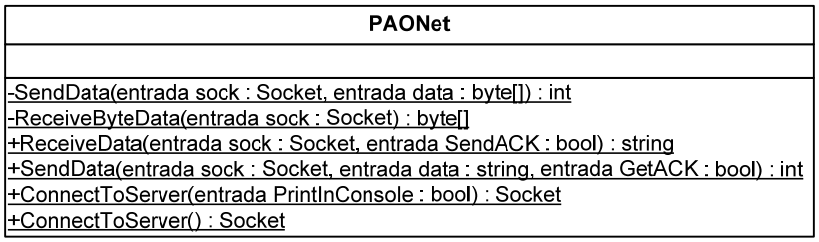


Figura 24 Diagrama UML de PAONet

La clase *PAOUtil* contiene utilidades para la generación de códigos aleatorios que se restrinjan a una política de seguridad determinada, así como los métodos que permiten pasar de estructuras o matrices en formato de bytes a la correspondiente cadena de caracteres. Entre sus funciones más importantes tenemos las siguientes:

- `GenerateUserPIN`, genera las claves aleatorias.
- `AddPasswordPolicy`, restringe las claves aleatorias generadas para que se adapten a una política de seguridad.
- `GetStringEncoding`, obtiene una cadena de caracteres a partir de una matriz de bytes.
- `GetByteEncoding`, obtiene una matriz de bytes a partir de una cadena de caracteres.

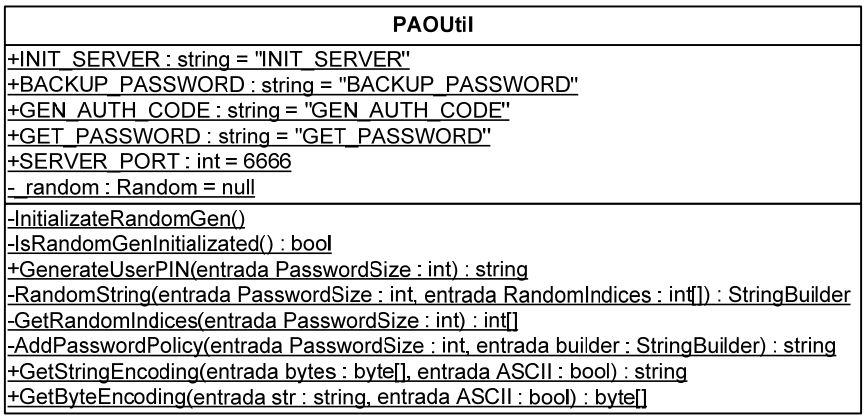


Figura 25 Diagrama UML de PAOUtil

La clase *PAOLogger* que únicamente posee un fichero, formatea los mensajes que se envían a un fichero de texto.

De esta librería no se muestra un diagrama UML ya que no existe interacción entre los elementos aunque es cierto que la clase *PAOUtil* se utiliza desde la clase de cifrado, *PAOCryptoUtil*, y desde la clase de acceso a la red, *PAONet*, para la conversión de cadena de caracteres a matriz de bytes y viceversa.

3.5 La aplicación PAOServer.exe

3.5.1 Descripción y funciones

La aplicación *PAOServer* es un servidor TCP/IP que realiza todas y cada una de las funciones del servidor de claves. Únicamente puede ser accedido desde localhost y debe estar instalado en cualquier servidor con acceso restringido. El servidor de claves puede realizar las siguientes operaciones:

- Realiza una copia de respaldo de la clave AES simétrica generada para un usuario con la que se cifró su par de claves RSA generadas y exportadas en formato XML.
- Genera un código de seguridad para que el usuario cliente sea capaz, a partir del conocimiento de este código, de recuperar la contraseña citada anteriormente.
- Obtiene la contraseña propiamente dicha a partir del código de seguridad comentado.

Es necesario destacar que el servidor debe inicializarse cada vez que la máquina donde reside se inicia. Esto es así debido a que este servidor requiere de una contraseña fuerte para inicializarse y descifrar el contenido de alguno de sus ficheros de configuración.

De esta contraseña no queda constancia ni por escrito ni en código en el servidor en texto claro y se liberará de la memoria justo después de su comprobación y utilización, que se detalla en este mismo apartado posteriormente.

3.5.2 Configuración

La configuración del servidor se basa en tres ficheros *paoserver.exe.config*, *setup.ini* y *users.pkz*. El archivo *.config* se encuentra en el mismo directorio que el ejecutable y la ruta de los otros dos ficheros es personalizable por el administrador junto con otras opciones que enumeramos a continuación en el fichero *.config*.

El fichero *paoserver.exe.config* tiene formato XML y mediante él se pueden especificar los siguientes parámetros:

- `SERVER_PORT`: 6666 por defecto, puerto en el que el servidor escucha las peticiones.
- `NUM_DAYS_EACH_SYNC`: 1 por defecto, especifica el número de días entre los cuales se realiza una sincronización entre el contenido en memoria y el fichero de respaldo users.pkz
- `NUM_DAYS_EACH_PASSWORD_CHANGE`: 7 por defecto, especifica el número de días en los que cambia la contraseña simétrica utilizada para cifrar el resto de contraseñas de los usuarios de la red.
- `AUTH_CODE_RANDOM_SIZE`: 10 por defecto, especifica la longitud del código aleatorio que se le genera a un usuario para que pueda obtener la clave simétrica que guardó en el servidor de claves.
- `NUM_OF_MINUTES_CODE_VALID`: 30 por defecto, tiempo durante el cual se considera válido el código aleatorio que se genera.
- `SETUP_PATH`: cadena vacía por defecto, ruta al fichero setup.ini.
- `USERS_PATH`: cadena vacía por defecto, ruta al fichero users.pkz.
- `RSA_KEY_SIZE`: 2048 por defecto, tamaño de la clave RSA del servidor. Una vez creados los ficheros este parámetro no debe cambiarse.

El fichero *setup.ini* contiene el vector de inicialización que junto con el SHA-256 de la contraseña del administrador forman constituyen los parámetros necesarios para cifrar AES-256, un par de claves RSA generadas para el correcto funcionamiento del servidor cifradas AES a partir de la clave deducida anteriormente y la contraseña a partir de la que se deduce la clave cifrada RSA con el certificado anterior. El exceso de seguridad en este fichero se debe a que este fichero es crítico para el mantenimiento del servidor de claves. Cada uno de los tres textos cifrados se separa por el carácter ':' en texto claro.

El fichero *users.pkz* contiene la clave simétrica utilizada para cifrar todas las claves simétricas utilizadas para cifrar las claves RSA que los usuarios tienen en su dispositivo de almacenamiento seguro, que debe recordarse, se almacena en el servidor ADAM para que en caso de extravío el dispositivo pueda clonarse. Además el fichero contiene los pares usuario-contraseña separados por ':'. Cada par de usuario contraseña se separa por el carácter ';'. Un ejemplo del fichero con tres usuarios añadidos, en texto claro, puede observarse a continuación:

CLAVE_Y_VI_AES256:USER1;CONTRASEÑA1:USER2;CONTRASEÑA2:USER3;CONTRASEÑA3

El sistema de cifrado del fichero *users.pkz* es similar al EFS (*Encrypted File System* o, en castellano, Sistema Encriptado de Ficheros) de Microsoft en el que se guarda las contraseñas cifradas en la cabecera del fichero. La decisión de no cifrar asimétricamente todo el fichero se basa en la velocidad de procesamiento fundamentalmente ya que se llegaría a un sistema ineficaz en el momento en el que el número de usuarios creciese. La siguiente imagen muestra gráficamente lo descrito con anterioridad particularizado para el sistema de ficheros de Microsoft (24):

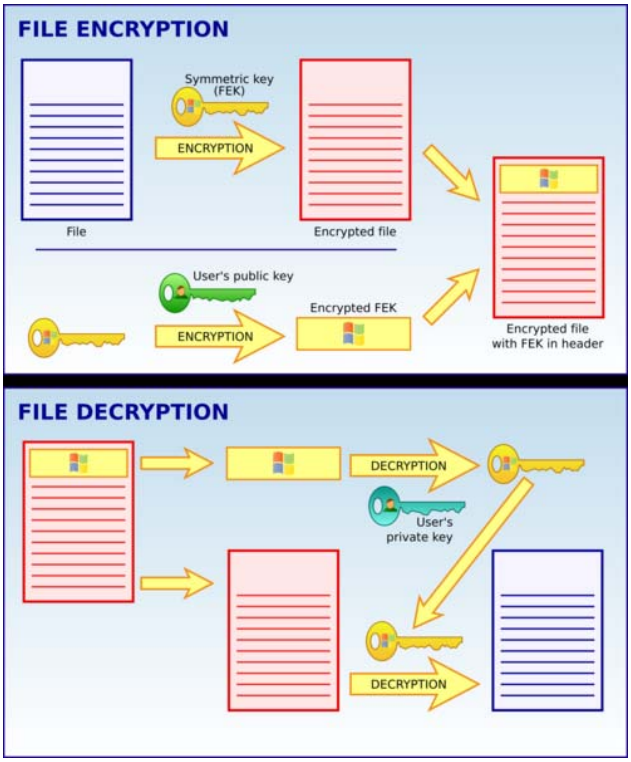


Figura 26 EFS

3.5.3 Funcionamiento

El servidor debe iniciarse o bien sin parámetros o bien especificando un solo parámetro que puede ser *verbose* o *v* y que indica si el servidor debe ejecutarse con la facultad de registrar y auditar todos los comandos que recibe el servidor o no. Para la depuración de la aplicación se ha hecho uso del paquete *System.Diagnostic* de Microsoft. El comando puede describirse de una manera más estándar como sigue:

```
> paoserver [verbose|v]
```

Una vez ejecutado el comando el servidor no queda operativo hasta el momento en el que el servidor reciba el comando INIT_SERVER. Mediante este comando el administrador especifica la contraseña inicial necesaria para arrancar plenamente el servidor. El administrador puede utilizar la consola PAOWS comentada en el punto posterior del documento para mandar comandos al servidor o puede desarrollar otra que se ajuste más a sus necesidades.

Una vez que la contraseña es conocida por el servidor se genera un hash SHA-256 de dicha contraseña y se accede al fichero *setup.ini* donde junto con el vector de inicialización en texto claro se crea el objeto capaz de realizar cifrados y descifrados simétricos AES o Rijndael. Una vez creado el objeto se descifra el par de claves RSA exportadas en formato XML y se crea el objeto con la capacidad de realizar cifrados y descifrados asimétricos; en este punto se descifra la parte final. En caso de no coincidir el proceso se considera como no satisfactorio y el servidor no es capaz de procesar comandos adicionales hasta que el proceso de inicialización no concluya con éxito.

Una vez realizada la fase inicial, el proceso de inicialización continúa creando los hilos de sincronización con la memoria física del ordenador y de cambio de contraseña. El cambio de contraseña fuerza, siempre, una sincronización del contenido de la memoria interna del ordenador al fichero *users.pkz*. La sincronización de dicho contenido consiste, básicamente, en crear un nuevo fichero *users.pkz* a partir del contenido en memoria. Primero se genera una nueva clave simétrica y luego se cifra todo el contenido como se describe en el apartado de configuración. A continuación se cifra la nueva clave simétrica con el certificado y todo se guarda en un fichero temporal, según se comenta en el apartado de configuración. En caso de resultado satisfactorio, se sobrescribe por el fichero original.

Una vez finalizadas estas operaciones, se realiza la parte final del proceso de inicialización del servidor, el descifrado del fichero *users.pkz*. Se descifra la última clave y vector de inicialización que se utilizaron para cifrar el fichero y una vez obtenida se guardan todos los pares usuario-contraseña en una estructura de datos para su acceso más rápido. Es en este punto exactamente cuando se considera que el servidor de claves está preparado para recibir comandos adicionales y que ha sido correctamente inicializado.

El proceso puede observarse en la figura siguiente.

Figura 27 Proceso de arranque PAOServer

Una vez realizado el proceso de inicialización el servidor puede ser capaz de procesar los comandos explicados en el apartado 3.5.1 dentro de este mismo punto, que son, realizar la copia de respaldo de una contraseña de usuario, obtener dicha contraseña y generación de códigos aleatorios para la obtención de las contraseñas.

Las copias de seguridad se realizan de forma directa, es decir, sin autenticación del usuario que lo realiza. Debido a la limitación de que el servidor únicamente está activo para conexiones desde la propia máquina no se considera el sitio adecuado para realizar dicha comprobación. En caso de que el usuario ya tuviera una clave guardada ésta se sobrescribe. Es importante remarcar que en este punto el usuario se considera totalmente autenticado y únicamente dicho usuario alcanza este punto y el caso de sobrescrito solamente se produce por robo o extravío del dispositivo de almacenamiento que poseyese.

La generación de códigos aleatorios debe ser realizada por el CAU cuando reciban una incidencia indicando que un usuario quiere clonar su dispositivo para lo que requiere la contraseña almacenada en el servidor. Dicho código aleatorio debe cumplir las siguientes propiedades, idénticas a las especificaciones que debía cumplir el PIN del dispositivo de cada uno de los usuarios:

- Longitud, la especificada en `AUTH_CODE_RANDOM_SIZE` aunque debe ser un valor no excesivamente largo para que no sea pesado en exceso el proceso tanto para el usuario como para el administrador.
- Utilización de al menos un símbolo no alfanumérico como @, #, ., &, etc.
- Existencia de al menos una letra minúscula, otra mayúscula y un número.

Como se ha especificado anteriormente dicho código únicamente es válido durante el tiempo especificado en el fichero de configuración, `NUM_OF_MINUTES_CODE_VALID`, 30 por defecto. Pasado dicho período de tiempo el código generado no es válido y el usuario recibirá un error describiendo la situación en la que se encuentra. En esta situación el usuario deberá establecer de nuevo contacto con el CAU y realizar el proceso de generación de código desde el principio.

Una vez que el usuario tiene un código aleatorio válido y está autenticado en su equipo, el usuario únicamente deberá especificar dicho código y el dispositivo se clonará

automáticamente siendo capaz de acceder a los mismos sitios a los que accedía anteriormente, inutilizando funcionalmente además el dispositivo anterior.

Las funcionalidades de crear una copia de seguridad y de obtener la contraseña AES son expuestas a los clientes finales mediante un servicio web que describe en el punto 3.7 y que es el que realiza la autenticación de los usuarios.

3.5.4 Implementación

Esta aplicación únicamente tiene una clase con el método de entrada o primer punto de ejecución llamada *PAOServer*. Dicha clase contiene una definición para todas las operaciones que el servidor TCP/IP es capaz de ejecutar. Las principales funciones se citan a continuación:

- `RSA_KEY_SIZE`, número de bits de las claves RSA del servidor.
- `NUM_DAYS_EACH_SYNC`, número de días que debe transcurrir para que el servidor realice una sincronización del contenido de memoria RAM a la memoria física del ordenador.
- `NUM_DAYS_EACH_PASSWD_CHANGE`, número de días que transcurre hasta que se cambia la clave simétrica del servidor.
- `AUTH_CODE_RANDOM_SIZE`, tamaño de caracteres del código de autorización para recuperar una clave.
- `NUM_OF_MINUTES_CODE_VALID`, número de minutes que el código de autorización es válido.
- `LoadFromSetupIni`, función que recupera toda la información del fichero de configuración, es decir, la última clave simétrica que se utilizó para cifrar el contenido del fichero, por defecto *users.pkz*.
- `CreateUsersFile`, crea el fichero de usuarios.
- `SyncFromMemory`, realiza un volcado del contenido de la estructura de datos a la memoria física del ordenador.
- `SyncFromFile`, copia en memoria RAM las claves simétricas de los usuarios contenidos en el fichero de usuarios cuando el servidor es inicializado.
- `ChangeSymmetricKey`, cambia la clave simétrica, volviendo a cifrar todo el fichero.
- `BackupPassword`, realiza la copia de respaldo de una clave simétrica de un usuario de la red corporativa.
- `GetPassword`, obtiene la clave simétrica de un usuario si su código de autorización es válido.
- `GetAuthCode`, genera un código de autorización que permite a un usuario recuperar su clave simétrica.

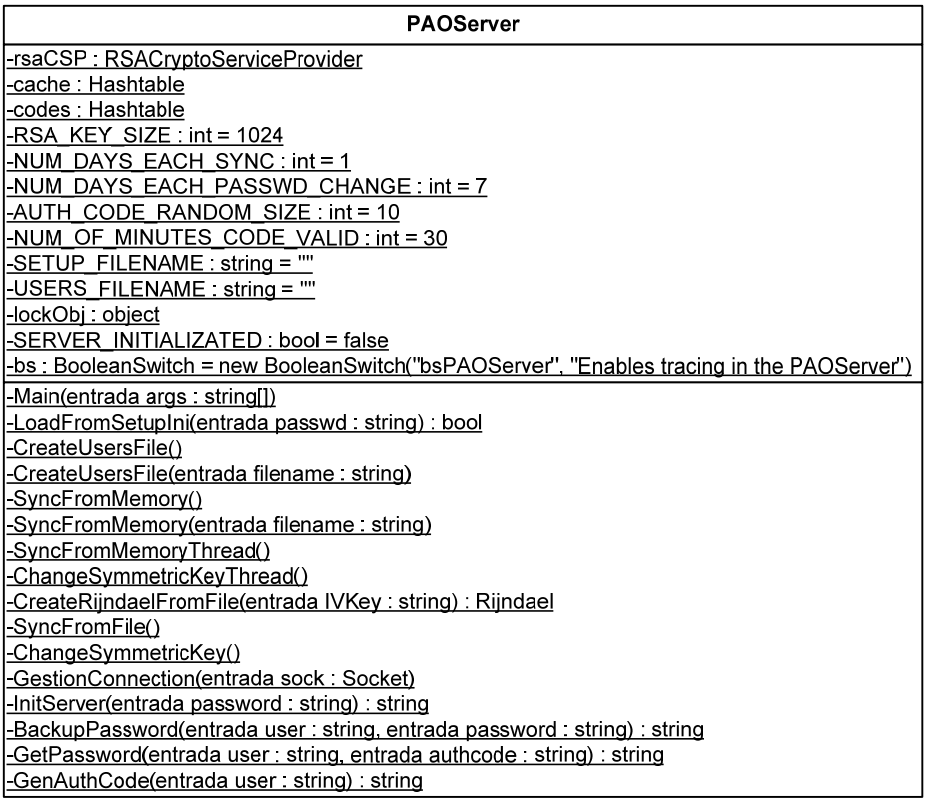


Figura 28 Diagrama UML de PAOServer

Al tratarse de una única clase no se especifica diagrama de bloques UML.

3.6 La aplicación PAOWS.exe

3.6.1 Descripción y funciones

La aplicación PAOWS se utiliza fundamentalmente para interactuar con el servidor de claves detallado en el apartado anterior. Las funciones que puede realizar se enumeran a continuación:

- Iniciar/Parar el servidor de claves.
- Crear los ficheros *setup.ini* y *users.pkz* por primera vez.
- Cambiar la contraseña con la que se cifra el certificado del servidor de claves.
- Mandar los comandos al servidor de realizar copia de respaldo, generar el código aleatorio y obtener la clave de un usuario especificando los parámetros necesarios que se detallaron en el punto anterior.

La invocación de la aplicación se realizará por consola de comandos y puede escribirse como se cita a continuación, si bien dependiendo del comando algunas opciones serán siempre obligatorias:

```
> paows create|start|init|close|stop|backup|gencode|getpwd|changemasterpwd [-
u user] [-pwd password] [-v|-verbose] [force] [-g gencode] [-p server_port]
[-key user_key] [-opwd pwd]
```

3.6.2 Escenarios de ejecución

En cualquier circunstancia que implique la comunicación con el servidor se puede especificar la opción –p para indicar el puerto en el que está escuchando el servidor. Si no se especifica se entiende que el servidor recibe peticiones en el puerto por defecto de la aplicación, 6666. En los escenarios en los que se ejecuten los comandos start|init|backup|gencode|getpwd puede especificarse dicho parámetro, en cualquier otro escenario su inclusión es, simplemente, ignorada, así como la inclusión de parámetros adicionales no determinantes para dicha operación.

La inicialización del servidor de claves por primera vez se basa fundamentalmente en la creación de un certificado de 2048 bits por defecto, aunque como se comentó en el apartado anterior, esta opción es modificable a partir del fichero de configuración del mismo, para el cifrado de datos y la generación de la primera clave simétrica a utilizar. La clave AES-256 se deriva a partir del SHA-256 de una contraseña de usuario que es la que se utilizará a partir de ese momento para la inicialización del servidor de claves. Dicha contraseña no se almacena en ninguna parte de manera física. Como resultado de la operación de inicialización de la plataforma se crea el fichero *setup.ini*. El comando para crear los ficheros en el servidor es:

```
> paows create –pwd contraseña [force]
```

La opción *force* es opcional y la contraseña, especificada por –p, será la contraseña que de ahí en adelante se utilizará para arrancar el servidor. La opción *force* se especifica a la hora de sobrescribir el fichero *setup.ini* y crear un nuevo certificado de servidor. Esta opción se debe confirmar durante el proceso ya que si se crea un nuevo fichero *setup.ini*, el fichero *users.pkz* que contiene todas las claves de todos los usuarios queda totalmente inservible.

Para arrancar el servidor se deben especificar los comandos *init* o *start* ambos obtienen el mismo resultado y ejecutan las mismas partes de código por lo que se utilizan indistintamente. Únicamente se debe especificar la contraseña que se utilizó para crear el fichero setup.ini. La aplicación lanza el proceso *paoserver* en modo desasistido y, una vez arrancado el proceso, envía al servidor el comando `INIT_SERVER` descrito en el punto anterior. La ejecución de la aplicación para el arranque del servidor se puede observar en la siguiente línea:

```
> paows start|init –pwd contraseña [-p port]
```

Para detener el proceso de servidor se ejecuta la aplicación con la opción *close* o *stop* indistintamente, sin especificar ningún parámetro adicional. Esta opción únicamente se realiza en las tareas de pruebas mientras la aplicación se encuentra en pre-producción, aunque no es necesaria ya que la aplicación servidor está preparada para responder a fallos puntuales de la máquina en la que reside sin comprometer el contenido de los ficheros de configuración. Para ejecutar esta opción se usa:

```
> paows close|stop
```

Para realizar una copia de seguridad en el servidor de claves únicamente se debe especificar la opción de *backup* especificando las opciones *–u* y *–key*. Este comando es enviado al servidor por TCP/IP que realiza la operación indicando el resultado obtenido del mismo. En el parámetro *–u* indica el nombre del usuario, incluyendo dominio, del cual se quiere realizar una copia de su clave de seguridad mientras que el parámetro *–key* especifica la clave de seguridad o clave AES de la que se va a realizar una copia de respaldo. Para hacer uso de esta opción se debe ejecutar la siguiente línea en la consola de comandos:

```
> paows backup –u user –key user_key [-p port]
```

Como proceso previo a la recuperación de la clave AES el usuario debe ponerse en contacto con el CAU. Mediante este proceso un usuario administrador de sistema designado a tal efecto debe generar un código de autenticación o código de seguridad para que el usuario pueda realizar el proceso. Para realizar el descrito proceso el administrador debe ejecutar la aplicación especificando la opción *gencode* con la opción *–u* que le permite indicar el usuario para el que se crea el código de validación. Este comando es enviado al servidor TCP/IP que guarda internamente el usuario para el que se ha generado el código de validación, el instante de tiempo o huella temporal en la que el servidor ha enviado este comando y el código hash SHA del código de validación generado. Nótese que no queda constancia escrita del código de validación generado ya que la transmisión de dicho código se recomienda que se realice por vía telefónica entre el usuario y el administrador del sistema. Una vez el usuario ha obtenido el código de validación éste dispone de toda la información necesaria para realizar el proceso de obtención de su clave AES o, lo que es lo mismo, de realizar un clon exacto de su dispositivo de almacenamiento seguro. En caso de que el usuario ya tuviese asignado un código de validación, éste se sobrescribe sin importar si dicho código estaba o no activo cuando el proceso de creación del nuevo código se inicia. Para la generación de dicho código debe ejecutarse el siguiente comando en el terminal:

```
> paows gencode –u user [-p port]
```

Si el usuario requiere su contraseña AES debe poseer al menos el código de validación obtenido según el proceso descrito en el párrafo anterior. Dicha clave AES es requerida cuando se está realizando una copia exacta del dispositivo de seguridad que un usuario poseía y ya sea por extravío, robo u olvido, no dispone en el momento de su utilización. El relativamente tedioso proceso de tener que realizar una llamada de teléfono para generar un código de validación no responde tanto a un aumento importante de la seguridad (ya que la seguridad del sistema aumenta aunque no de manera importante) sino a forzar al usuario a realizar un proceso que considere cansino o, que repetido en demasía, puede producir un cierto efecto de rechazo en el usuario de manera que el número de olvidos y de extravíos disminuya. En cualquier caso, una vez que el usuario dispone de su código de validación es capaz de recuperar su contraseña; esta capacidad está expuesta junto con la de realizar una copia de respaldo tal y como se describe en el siguiente punto. Para la recuperación de la contraseña AES de un usuario debe ejecutarse el comando que se cita a continuación siendo consciente de las restricciones temporales impuestas por la configuración del entorno por la cual un usuario no es capaz de recuperar su contraseña AES `NUM_OF_MINUTES_CODE_VALID` minutos después de la generación de su código de validación:

```
> paows getpwd –u user –g gencode [-p port]
```

La última de las funcionalidades de la aplicación debe usarse con moderación y precaución. Esta opción se refiere a la capacidad del administrador(es) delegado(s) al mantenimiento de sistema de cambiar la clave utilizada en cada inicio del sistema o clave que se especifica en el parámetro *pwd* cuando se utiliza la opción start|init. Dicho cambio de contraseña presupone el proceso de la siguiente figura.

Figura 29 Proceso de cambio de contraseña maestra

Si el proceso de inicialización del servidor de nuevo se produce con éxito, entonces las copias de respaldo de los ficheros de configuración se eliminan del sistema no quedando constancia del cambio de contraseña. En caso de que la inicialización del servidor de seguridad no se realice con éxito, el servidor se cierra de nuevo se sustituyen los nuevos ficheros creados por los de la copia de seguridad y se elimina la copia de seguridad del sistema. Se recomienda utilizar este proceso fuera del horario laboral de los usuarios en la medida de lo posible ya que, como puede observase, conlleva una denegación de servicio a los usuarios del sistema. Para realizar este proceso debe ejecutarse la siguiente línea de comando:

```
> paows changemasterpwd –pwd contraseña –opwd contraseña_antigua [-v|-verbose]
```

Por tanto, no es necesario que el servidor de seguridad o servidor TCP/IP esté arrancado ya que en caso de estarlo, se para. Se deben incluir ambas contraseñas, la indicada en el parámetro *–opwd* para el acceso al certificado antiguo y la especificada por el parámetro *–pwd* para la creación del nuevo fichero *setup.ini* como fichero maestro de la configuración del servidor TCP/IP de claves. Cuando el proceso finaliza el servidor TCP/IP se arranca de nuevo con la nueva contraseña, habilitando o no la depuración según especifique el parámetro *verbose*.

No se ha estimado como necesario ofrecer al administrador el cambio del certificado de servidor ya que la seguridad ofrecida por AES-256 es considerada como máxima seguridad por el NIS y el certificado es suficientemente potente para que no deba ser cambiado, por defecto, RSA-2048.

3.6.3 Descripción del código fuente

La aplicación PAOWS como se ha explicado anteriormente es una herramienta por consola de comandos para la ejecución de los servicios publicados por la arquitectura, así como la creación y tratamiento del fichero de configuración del servidor, por defecto, *setup.ini*. A continuación se detallan las funciones más importantes de la librería:

- `RSA_KEY_FILE`, tamaño de la clave RSA que debe coincidir con el valor de clave especificado en el apartado *Descripción del código fuente* de la aplicación PAOServer.exe.
- `CreateSetupIniFile`, crea el fichero de configuración para la correcta inicialización y mantenimiento de los datos de manera segura en el servidor.
- `StartServer`, inicializa el proceso de la aplicación del servidor seguro en segundo plano.
- `CloseServer`, finaliza el proceso de la aplicación del servidor seguro.
- `GetAuthCode`, obtiene un código de autorización del servidor por petición directa, no mediante los servicios web publicados.
- `BackupPassword`, realiza una copia de respaldo de una clave simétrica de un usuario directamente sobre el servidor seguro sin hacer uso de la funcionalidad expuesta por los servicios web.
- `GetPassword`, obtiene una contraseña de usuario del servidor seguro para un usuario especificando su código de autorización.
- `ChangeMasterPwd`, cambia la contraseña maestra del fichero de inicialización del servidor.

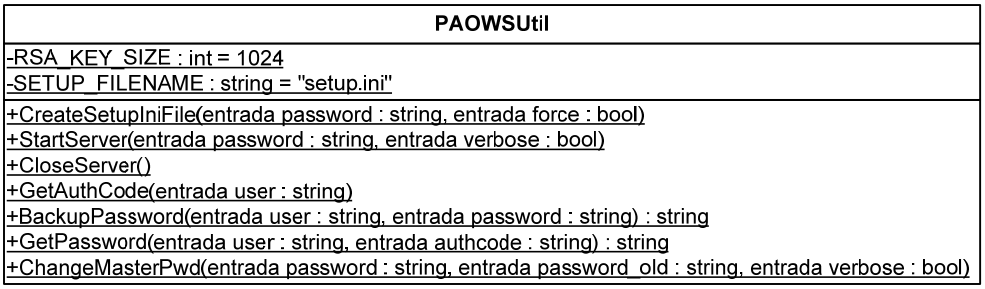


Figura 30 Diagrama UML de PAOWSUtil

3.7 El servicio web pao.asmx

3.7.1 Descripción y funciones

El servicio web *pao.asmx* permite a la librería distribuir de manera independiente de la plataforma el acceso a los procesos que el servidor de claves puede realizar. Es importante remarcar el hecho de que sea independiente de la plataforma ya que aunque el servicio web ha sido programado en C#, éste puede ser consumido desde cualquier cliente. En este proyecto el único cliente programado para la realización de las pruebas es para plataformas Windows, debido a las especificaciones del cliente que quería integrar el desarrollo en una aplicación Windows ya realizada. El servicio web expone las capacidades del servidor de claves de realizar un proceso de copia de seguridad de la clave AES de los usuarios en el entorno y la

posibilidad de los usuarios de recuperar esa clave previamente guardada. Las operaciones reciben el siguiente nombre en código:

- `BackupPassword`
- `GetPassword`

La función *BackupPassword* recibe por parámetros el usuario y la contraseña respectiva al mismo y devuelve una cadena de caracteres que indica el resultado del proceso, que puede variar entre las siguientes posibilidades:

- *"ERROR: Please contact CAU to inform that Server has not been properly started"*. En caso de que no se haya podido establecer conexión con el servidor de seguridad o bien el servidor haya sido arrancado, pero no correctamente inicializado, esto es, que el administrador de sistema no haya provisto la clave necesaria para descifrar el certificado de servidor.
- *"ERROR: A user is only able to work with his own password."*. En caso de que un usuario esté intentando guardar una contraseña para un usuario en el dominio diferente al que está autenticado en su máquina cliente en ese instante.
- *"OK"*. En caso de que la operación haya resultado satisfactoria.

Por otra parte la función *GetPassword* recibe el usuario y el código de autenticación por parámetros. Dicho código es requerido para obtener la contraseña de un usuario, está disponible durante 30 minutos por defecto y debe haber sido creado por un administrador de sistema válido en el dominio. Como resultado de la operación se puede obtener una de las siguientes cadenas de caracteres:

- *"ERROR: Please contact CAU to inform that Server has not been properly started"*. En caso de que no se haya podido establecer conexión con el servidor de seguridad o bien el servidor haya sido arrancado, pero no correctamente inicializado, esto es, que el administrador de sistema no haya provisto la clave necesaria para descifrar el certificado de servidor.
- *"ERROR: No valid data available. Please contact CAU again"*. En caso de que los datos generados por la aplicación no sean válidos, es decir, no exista un par de huella temporal y de clave de autenticación válida para el usuario que realiza la petición.
- *"ERROR: No valid data available. The gencode has expired. Please start the proccess again."*. En caso de que los datos registrados para el usuario por el entorno sean válidos, pero el proceso se haya denegado porque el usuario ha tardado más del máximo de tiempo permitido en recuperar su clave.
- *"ERROR: Key is not available or code has not been provided by CAU"*. En caso de que el usuario no haya guardado previamente su contraseña en el servidor.
- *"ERROR: A user is only able to work with his own password."*. En caso de que un usuario esté intentando obtener una contraseña para un usuario en el dominio diferente al que está autenticado en su máquina cliente en ese instante.
- En caso de que el proceso de obtención de clave se haya realizado con éxito el servicio web responderá con la clave en formato de cadena de caracteres hexadecimales.

El modo en el que el servicio web realiza las operaciones expuestas se basa fundamentalmente en la ejecución de la aplicación PAOWS tal y como se comentó en el punto anterior ejecutando como se describe a continuación:

- Para la operación *BakcupPassword*: `> paows.exe –u user –key user_key`
- Para la operación *GetPassword*: `> paows.exe –u user –g gencode`

Visto esto es obvio que el servicio web no es más que una interfaz distribuida y multiplataforma para el acceso a las funciones del servidor de seguridad para la copia de respaldo y la recuperación de las contraseñas.

Debido a que el proceso requiere el envío de contraseñas por la red en texto plano se requieren dos procesos de manera que la seguridad se mantenga íntegra:

- Frente a la réplica de paquetes y envío de datos cifrados el servicio web se expone en los clientes a través de una interfaz SSL de manera que los datos viajan cifrados con un certificado de servidor. Véase que se requiere un certificado de servidor y que éste es independiente del mecanismo de cifrado de claves que el servidor posee en sí y que se basa en un par de claves RSA-2048, por defecto. SSL requiere un certificado válido por una autoridad de certificación de confianza en todos los clientes de manera que no requiera instalaciones adicionales en los navegadores o sistemas operativos de los clientes, por lo que el par de claves automáticamente generadas no es válido.
- Frente a la posibilidad de que un usuario guarde o recupere una contraseña en nombre de otro usuario se utiliza un escenario de seguridad integrada en el dominio de manera que todos los usuarios que accedan al servicio web tengan un *token* de seguridad válido en el dominio o sean capaces de crear uno para el servicio web a partir de un usuario y una contraseña válidos en el dominio.

En este punto resta únicamente responder a la pregunta de que si un usuario no es capaz de recuperar o realizar una copia de respaldo de una contraseña en nombre de otro usuario, cuál es entonces el motivo de que los servicios web admitan un parámetro *usuario* en sendas operaciones. La respuesta es que en este entorno dicho parámetro es innecesario, pero para futuras ampliaciones en las que ciertos grupos de dominio puedan realizar operaciones sobre otros usuarios dotándoles así de un perfil de administrador en el entorno, este cambio no requiera la actualización en los entornos clientes anteriores y se les pueda dotar de dichas funcionalidades únicamente mediante anexión al comentado grupo de dominio.

4 Escenarios de Pruebas

Este apartado explicita las pruebas realizadas sobre la API para su puesta en producción. Dichas pruebas requieren la creación de un cliente que utilice la API. Así las futuras implementaciones que otros clientes realicen y que hagan uso de esta API podrán asegurar que, en caso de error, el error se ha producido dentro de su propio código y no dentro del entorno de trabajo o dentro de las librerías desarrolladas.

La API se ha realizado en código administrado por lo que la memoria que se reserva o libera es responsabilidad del recolector de basura del entorno de ejecución por lo que las pruebas se basarán en el correcto funcionamiento de la API bajo las circunstancias descritas en cada una de las mismas. Este correcto funcionamiento puede, en algunos casos, lanzar excepciones que deban ser capturadas por la aplicación haciendo uso de la API. Por otro lado todas las pruebas realizadas sobre el servicio web o el servidor TCP/IP o cualquiera de las aplicaciones realizadas deben estar libres de errores ya que son aplicaciones finales en sí.

En cada uno de los apartados se realiza una descripción de la situación de prueba, el resultado que se debe obtener de dicha prueba y si se ha obtenido un resultado parcial o completo o si el resultado obtenido no es el esperado. Al no tratarse de procesos pesados o que interactúen con otros sistemas en tiempo real los tiempos de cómputo no se incluyen en el informe al no considerarse relevantes.

Por tanto, se especifica como requisito que la API o las aplicaciones desarrolladas para la administración o distribución del entorno sean capaces o no de responder de la manera adecuada en cada uno de los casos suponiendo que el entorno de ejecución sobre el que se realiza libera conscientemente la memoria y ejecuta las operaciones en un tiempo razonable en función de la carga y el tipo de ordenador cliente sobre el que se corra la aplicación cliente final.

4.1 Prueba de las diferentes funcionalidades

En este apartado se van a probar las diferentes funcionalidades del API del cliente. A pesar de que la API del cliente está desarrollado en C# las pruebas de la aplicación cliente se han desarrollado en su mayoría en VB 8.0 (VB.NET) que era el lenguaje en el que se iba a implantar por vez primera la API en una aplicación cliente. Durante los siguientes apartados se escribe el código del programa de prueba así como la salida que produce comentando si esta es la esperada o no según se ha descrito en el apartado anterior.

4.1.1 Creación de las claves

El siguiente programa de prueba muestra el código necesario a introducir en una aplicación cliente para crear las claves dentro del dispositivo de seguridad. En el caso de esta prueba la API no ha sido compilada de manera que esté integrada con el servicio web y, por tanto, no intentará realizar una copia de respaldo en el servidor de claves de la clave AES generada. En este caso concreto la máquina cliente ya dispone los *drivers* necesarios para interactuar con el dispositivo de seguridad que se va a utilizar, el Aladdin eToken PRO 32K 4.28.

Para demostrar la correcta creación de las claves se abrirá el dispositivo de almacenamiento seguro a partir de la función *OpenToken*, dicha función lanzará una excepción en caso de que

el contenedor por defecto no se encuentre. El nombre de este contenedor de claves corresponde con el siguiente formato:

PAO_DefaultContainer_Dominio_Usuario_Dia_Mes_Año_Hora_Minuto_Segundo

La inclusión de la fecha dentro del contenedor de claves será utilizada como se ha comentado anteriormente para comprobar que la fecha asociada para el usuario dentro del dispositivo de seguridad es la misma que la fecha asociada en el servidor de directorio del dominio. Además se comprueba que el usuario que ha realizado el login en el dominio y que está intentando acceder al dispositivo de seguridad es el mismo que creó el contenedor de claves dentro del dispositivo. Estas son comprobaciones que se realizan a la hora de hacer uso del dispositvo de seguridad.

El código para crear una nueva entrada dentro del contenedor de claves es el que se detalla a continuación:

```
'Prueba de creación de un nuevo par de claves RSA dentro del dispositivo de seguridad
'Obteniendo como resultado:
'    - La clave pública RSA
'    - El par de claves RSA, cifradas AES
'    - El PIN, cifrado RSA
'    - La fecha de creación del token
Dim token As PAOAPI.PAOToken
token = New PAOAPI.PAOToken()

Dim pkcert As String = ""
Dim tkDate As String = ""
Dim outCert As String = ""
Dim encodedPin As String = ""
Dim login As String = ""

login = token.CreatePAOToken(pkcert, tkDate, outCert, encodedPin)
token.CloseToken()
token.OpenToken(login, tkDate)
```

Como resultado de la prueba se muestran todos los mensajes de depuración contenidos en la librería desarrollada en C++ así como el nuevo PIN del usuario en el token, generado automáticamente.

Además puede observarse cómo se iniciailiza el dispositvo de almacenamiento seguro a partir de la contraseña que tiene por defecto y que viene de fábrica, esto es, 1234567890 y cómo se genera una contraseña aleatoria que cumpla las políticas del dispositivo. Todo esto puede verse en la siguiente figura:

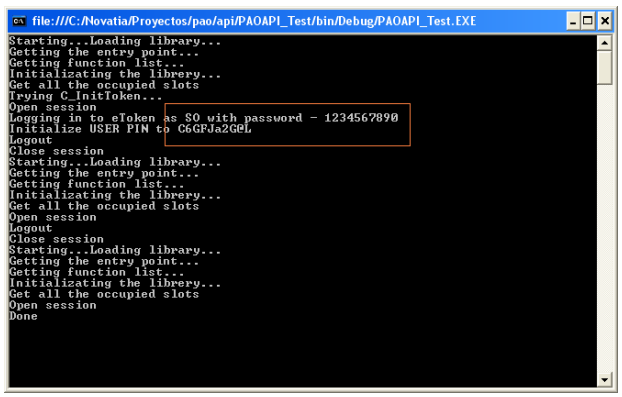


Figura 31 Salida: Creación de un nuevo token

En caso de que el proceso de creación de claves hubiese fallado la función *OpenToken* habría lanzado una excepción. Además se puede ver el contenedor de claves creado juanto con la clave RSA de 1024 bits en la siguiente figura obtenida a partir de la interfaz de Aladdin que se puede instalar con los *drivers* del dispositivo.

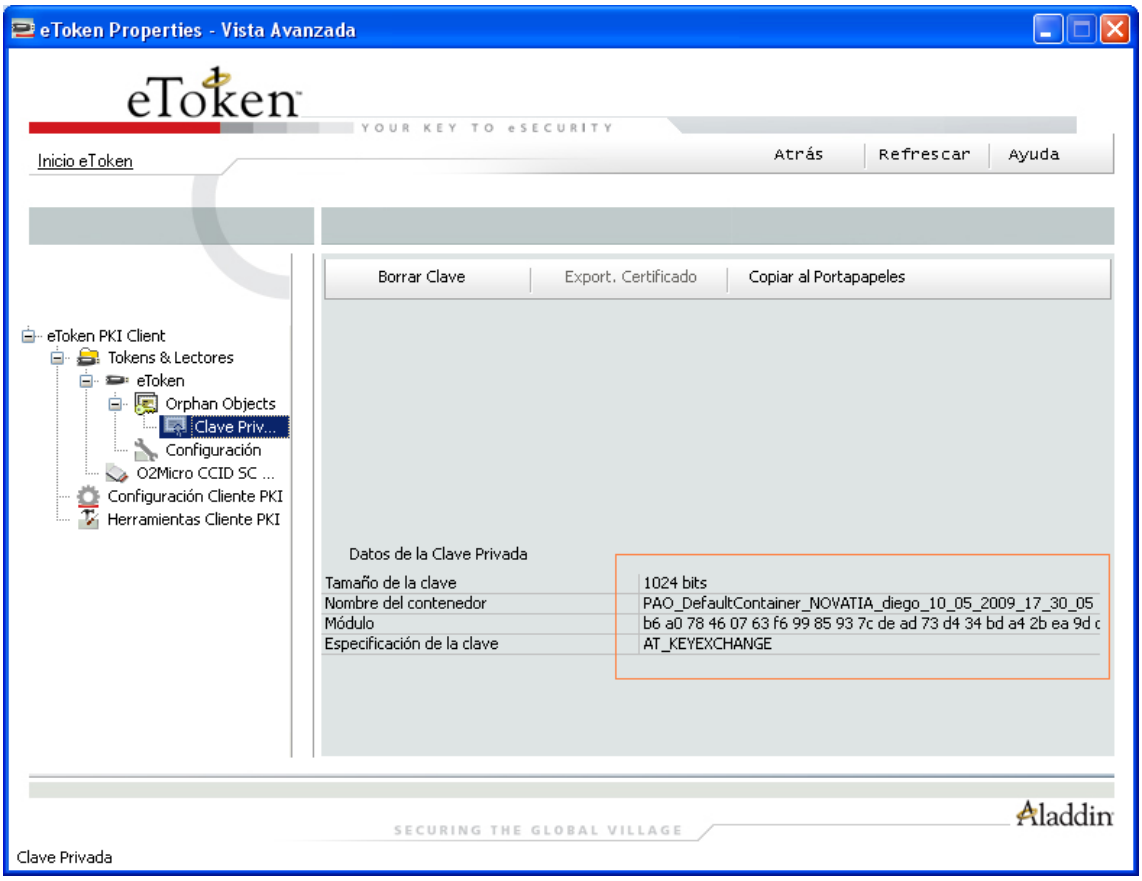


Figura 32 Aplicación Aladdin con el nuevo contenedor

4.1.2 Cambio de PIN

El siguiente código muestra el proceso de cambio del PIN del dispositivo de seguridad. La manera para probar que dicho cambio ha sido efectivo se realizará siguiendo los pasos descritos en la siguiente figura.

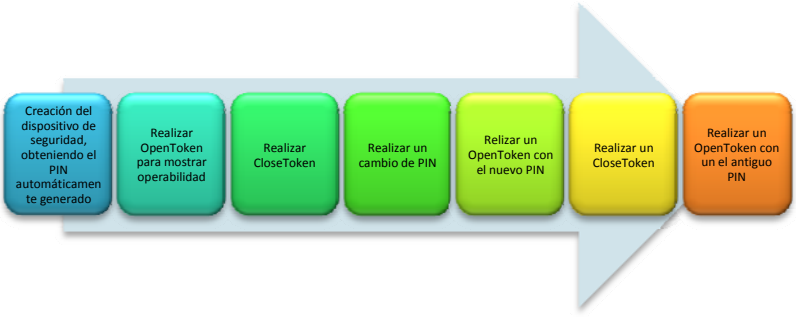


Figura 33 Proceso de cambio del PIN de usuario

El código que representa el proceso detallado en la figura anterior se muestra a continuación:

```
Dim token As PAOAPI.PAOToken
token = New PAOAPI.PAOToken()

Dim pkcert As String = ""
Dim tkDate As String = ""
Dim outCert As String = ""
Dim encodedPin As String = ""
Dim login As String = ""
Dim ErrorDesc As String = ""

login = token.CreatePAOToken(pkcert, tkDate, outCert, encodedPin)
token.CloseToken()
Console.WriteLine("Dispositivo de seguridad creado con fecha: {0}", tkDate)
token.OpenToken(login, tkDate)
Console.WriteLine("Cambiando PIN de usuario a pfincarrera@09")
If Not token.ChangeTokenPIN(login, "pfincarrera@09", encodedPin, ErrorDesc) Then
       Console.WriteLine("Error al cambiar el PIN " & ErrorDesc)
End If
Console.WriteLine("Proceso realizado con éxito")
token.OpenToken("pfincarrera@09", tkDate)
token.CloseToken()
token.OpenToken(login, tkDate)
Console.WriteLine("Done")
```

En este caso el proceso no finaliza imprimiendo la palabra *Done* por pantalla sino que lo hace lanzando una excepción propietaria cuyo mensaje es *User or PIN not valid*. Se puede ver este proceso en ambas figuras. La primera de ellas muestra los diferentes mensajes que se obtienen en la pantalla hasta que se produce la excepción que se muestra en la siguiente figura:

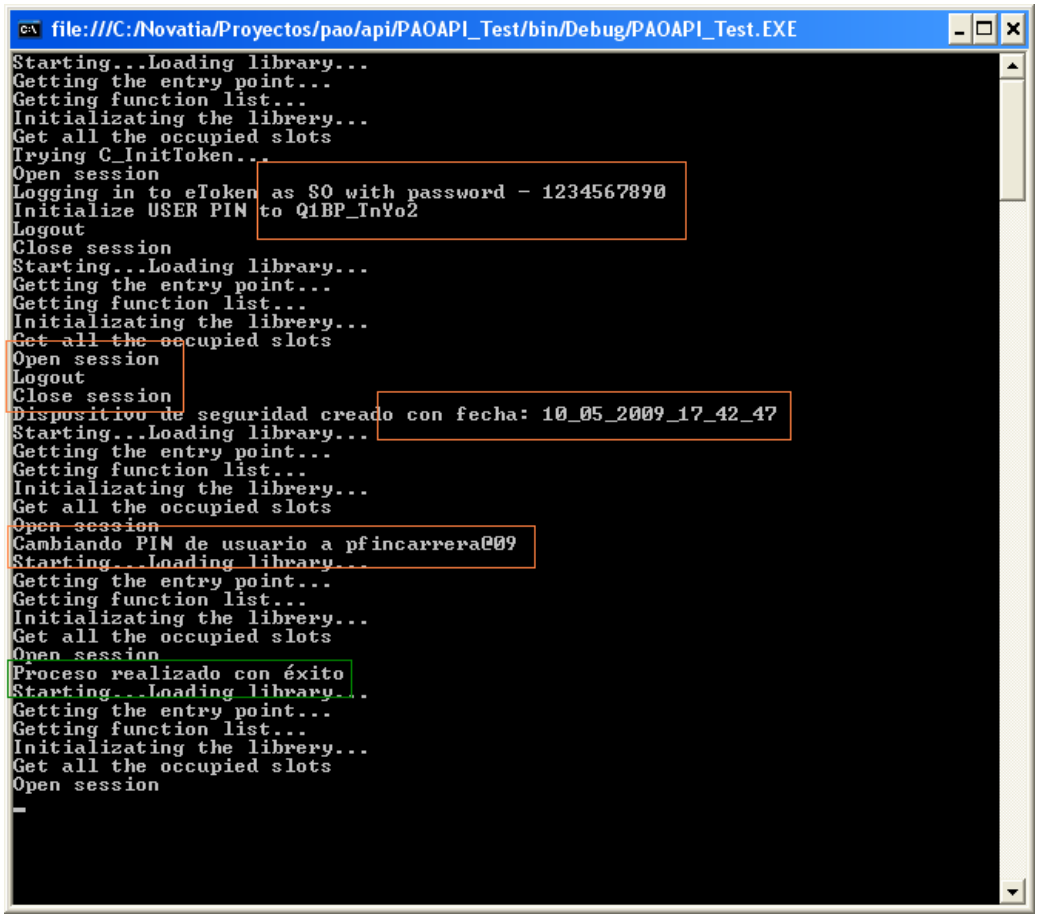


Figura 34 Salida: Cambio de PIN de usuario

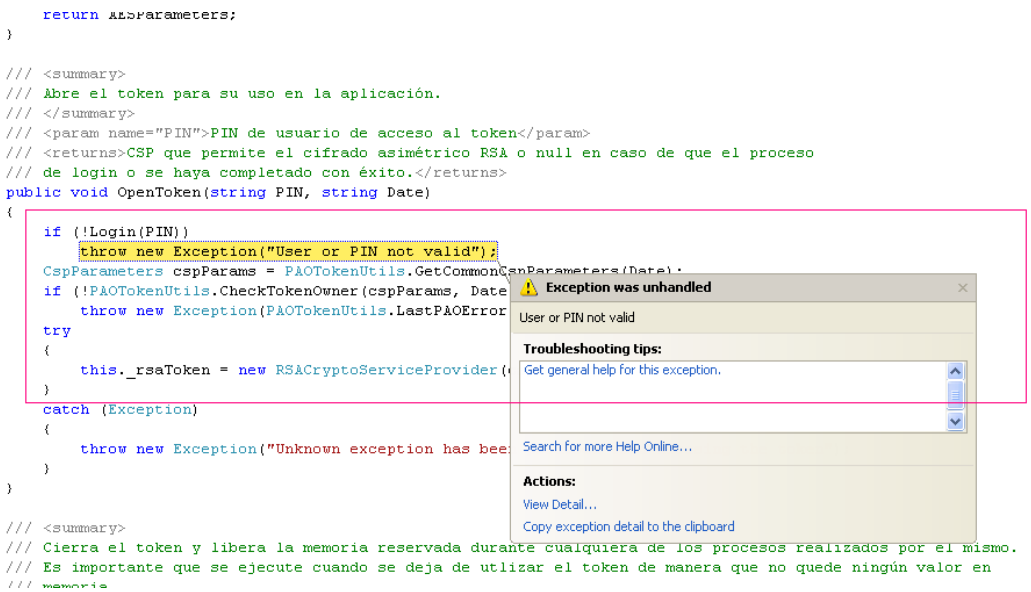


Figura 35 Salida: PIN no válido

La comprobación que se realiza cuando se ejecuta la función *ChangeTokenPIN* es para verificar que la nueva contraseña cumple las políticas de contraseñas guardadas en el dominio. Si se intenta cambiar por una contraseña que no cumple dichas especificaciones el proceso de cambiar el PIN del dispositivo de seguridad falla con el mensaje *"El nuevo password no cumple con las políticas de seguridad del token."*[4]. Esto se puede observar utilizando el siguiente código:

```
Dim token As PAOAPI.PAOToken
token = New PAOAPI.PAOToken()

Dim pkcert As String = ""
Dim tkDate As String = ""
Dim outCert As String = ""
Dim encodedPin As String = ""
Dim login As String = ""
Dim ErrorDesc As String = ""

token.OpenToken("pfincarrera@09", "10_05_2009_17_42_47")
Console.WriteLine("Cambiando PIN de usuario a pfincarrera@09")
If Not token.ChangeTokenPIN(login, "hola", encodedPin, ErrorDesc) Then
        Console.WriteLine("Error al cambiar el PIN " & ErrorDesc)
Else
        Console.WriteLine("Proceso realizado con éxito")
End If
```

[4] Este mensaje aparece en español porque es un mensaje que se le muestra al usuario final. El resto de excepciones se muestran en inglés porque en su mayoría son excepciones que se lanzan al administrador del sistema o a un fichero de log. El idioma en el que se lanza cada una de las excepciones fue requisito del cliente.

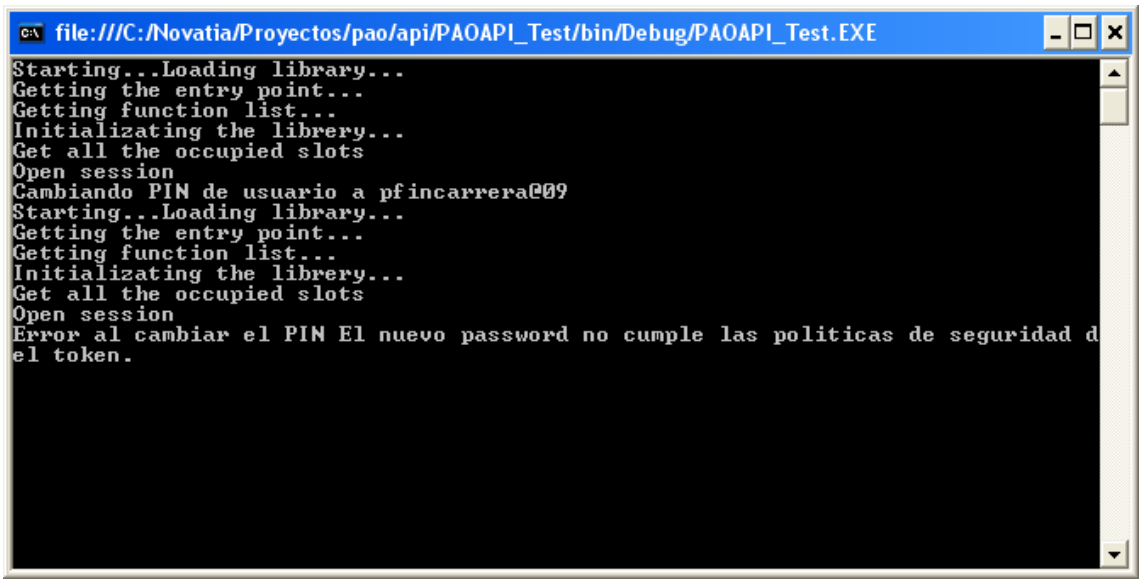


Figura 36 Salida: Error contraseña no cumple las políticas de seguridad

4.1.3 Cifrado asimétrico

Las siguientes pruebas muestran las capacidades de cifrado asimétrico como el propio título indica. En todos los casos se requiere de un dispositivo de almacenamiento de seguridad activo e inicializado. Aunque la librería puede realizar cifrados simétricos con AES y operaciones de hash con SHA, ambos en sus formatos de 256 bits, éstos no están expuestos para el uso de los clientes que adopten la API. El motivo fundamental es que el cifrado simétrico se basa en la clave AES que se almacena en el dispositivo de seguridad y que, o bien en el momento de la creación del dispositivo es conocida en la máquina cliente, o bien a través del código de validación, ésta se puede pedir al servicio web. Con las operaciones de hash el motivo es un tanto diferente, aunque se utilizan operaciones de hash sin contraseña, esta capacidad no se definió como fundamental para los clientes que utilicen la API y, por tanto, no fue expuesta.

La mejor manera de demostrar las capacidades de cifrado de la API es inicializando un dispositivo de almacenamiento seguro de Aladdin, cifrando un mensaje y descifrando el contenido cifrado para ver que el mensaje obtenido es exactamente idéntico al mensaje inicial. Esto es lo que se pretende en el siguiente bloque de código:

```
Dim ciphered As String
Dim deciphered As String
Dim token As PAOAPI.PAOToken = New PAOAPI.PAOToken()
ciphered = token.AsymmetricEncrypt("Mensaje de prueba")
Console.WriteLine(ciphered)
token.OpenToken("pfincarrera@09", "04_04_2009_19_30_11")
deciphered = token.AsymmetricDecrypt(ciphered)
Console.WriteLine(deciphered)
token.CloseToken()
```

Como se obtiene finalmente por pantalla el texto de “Mensaje de prueba” previo del mensaje cifrado RSA-1024 en formato hexadecimal. Esto se observa en la siguiente pantalla.

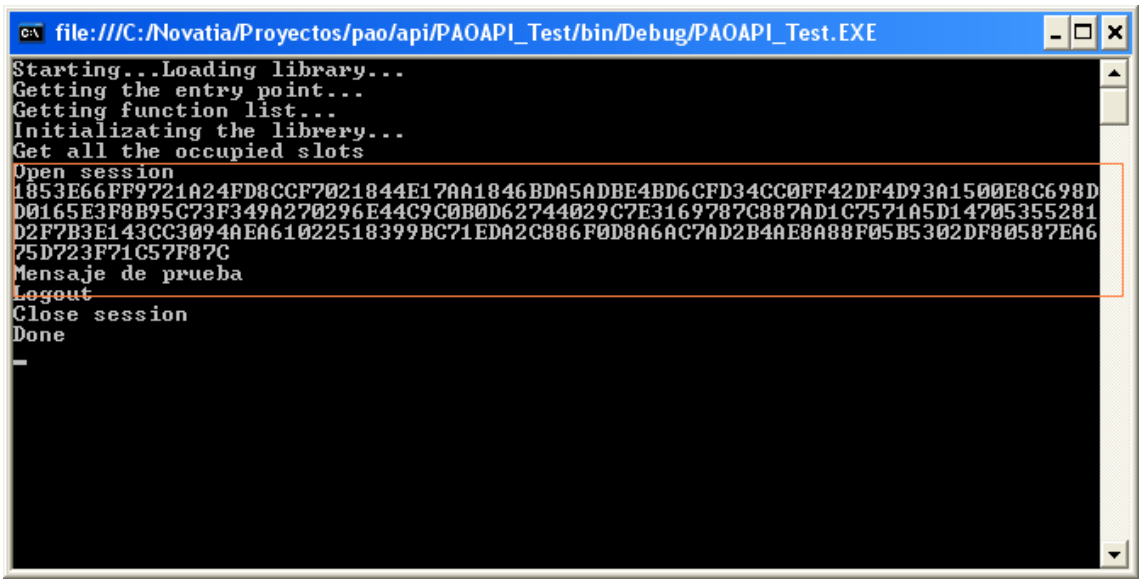


Figura 37 Salida: Mensaje cifrado RSA

En este caso se va a poner a prueba una de las propiedades del cifrado RSA: cifrando un texto de mayor longitud, va a observarse que la longitud del texto cifrado coincide con la anterior. El código utilizado para esta prueba puede observarse a continuación y el resultado se ve claramente en la siguiente figura:

```
Dim ciphered As String
Dim deciphered As String
Dim token As PAOAPI.PAOToken = New PAOAPI.PAOToken()
token.OpenToken("pfincarrera@09", "10_05_2009_17_42_47")
ciphered = token.AsymmetricEncrypt("Mensaje de prueba")
Console.WriteLine(ciphered)
deciphered = token.AsymmetricDecrypt(ciphered)
Console.WriteLine(deciphered)
ciphered = token.AsymmetricEncrypt("Mensaje de prueba Mensaje de prueba Mensaje de
prueba Mensaje de prueba")
Console.WriteLine(ciphered)
deciphered = token.AsymmetricDecrypt(ciphered)
Console.WriteLine(deciphered)
ciphered = token.AsymmetricEncrypt("M")        Console.WriteLine(ciphered)
deciphered = token.AsymmetricDecrypt(ciphered)
Console.WriteLine(deciphered)
token.CloseToken()
Console.WriteLine("Done")
```

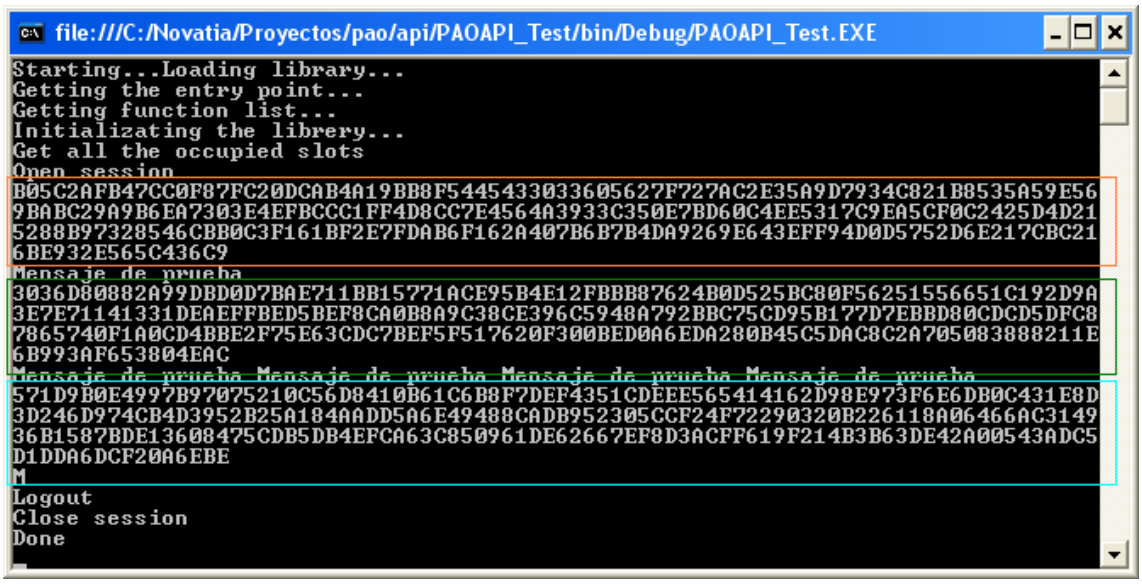


Figura 38 Mensaje cifrado RSA de longitud variable

4.1.4 Caché

La caché debe ser utilizada por el cliente propietario a su antojo. Esto quiere decir que la caché es una funcionalidad de la API, pero no define cuándo y dónde debe utilizarse, la caché únicamente almacena pares de usuario-contraseña para una o más aplicaciones, el resto son decisiones de la aplicación cliente.

Las pruebas con la caché se han limitado a la creación y modificación de entradas en un fichero de texto local del ordenador. Para permitir la delegación, entendida como diferentes pares de usuarios para una misma aplicación, las diferentes funciones de la librería buscan entradas en el fichero de texto almacenado de manera local y añaden o modifican las existentes según el diagrama de flujo expuesto en la Figura 18 Funcionalidad *AddAppToCache*Figura 18.

El siguiente bloque de código muestra cómo crear el fichero caché, crear y modificar entradas dentro de dicho fichero y cómo obtener los valores insertados dentro del fichero. El siguiente código muestra cómo realizar operaciones con la caché de la aplicación.

```
Dim token As PAOAPI.PAOToken = New PAOAPI.PAOToken()
token.OpenToken("pfincarrera@09", "10_05_2009_17_42_47")
token.CreateCacheFile(True)
token.AddAppToCache("Aplicacion001", "pfc001", "&&pFc&&001")
token.AddAppToCache("Aplicacion002", "pfc001", "&&pFc&&001")
token.AddAppToCache("Aplicacion003", "pfc001", "&&pFc&&001")
token.AddAppToCache("Aplicacion001", "pfc002", "&&pFc&&020")
token.AddAppToCache("Aplicacion001", "pfc001", "&&pFc&&005")
token.AddAppToCache("Aplicacion003", "pfc001", "&&pFc&&002")
token.AddAppToCache("Aplicacion002", "pfc001", "&&pFc&&002")
token.AddAppToCache("Aplicacion001", "pfc003", "&&pFc&&003")
token.AddAppToCache("Aplicacion003", "pfc005", "&&pFc&&002")

Dim hash As New Hashtable()
hash = token.GetLoginInfo("Aplicacion001")
Dim oEnumerador As IDictionaryEnumerator
oEnumerador = hash.GetEnumerator()
While oEnumerador.MoveNext()
       Console.WriteLine("User: {0} / Passwd: {1}", oEnumerador.Key, oEnumerador.Value)
End While
token.CloseToken()
```

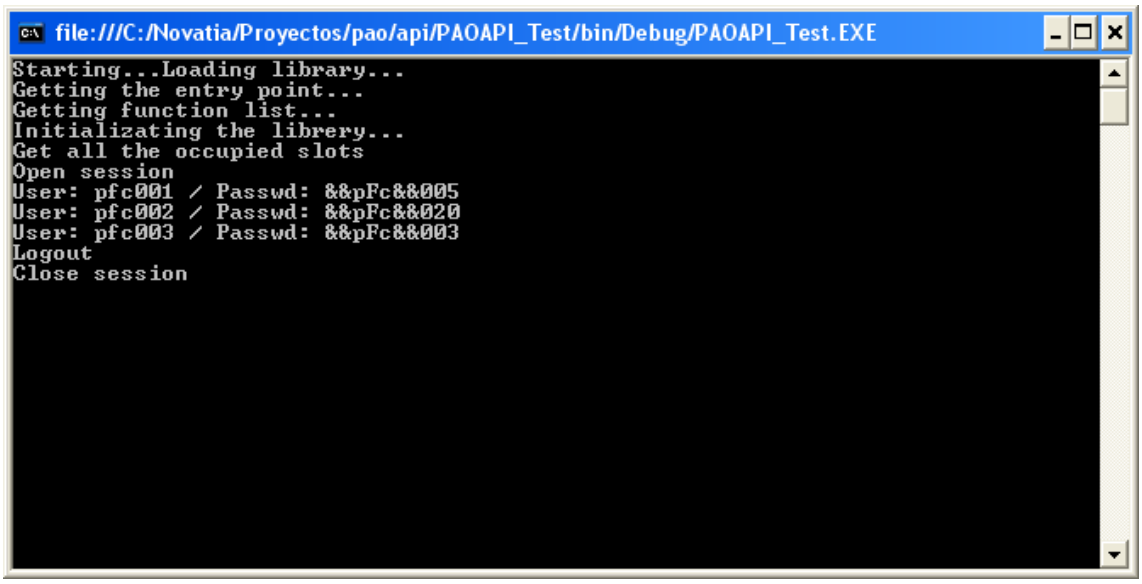


Figura 39 Salida: Interactuación con la caché

En el siguiente volcado de fichero se observa como el cifrado de los mismos usuarios y contraseñas da lugar al mismo volcado hexadecimal. Esto no se considera una brecha de seguridad ya que no aporta ninguna información nueva a la hora de realizar ataques de fuerza bruta y aunque el fichero contiene además la clave pública estos datos no suponen información adicional a la publicada en el directorio. En cualquier caso, al situar la ruta por defecto del fichero dentro de la carpeta de usuario, ésta debe ser accesible únicamente para usuarios administradores y para el propio usuario, remarcando una vez más que la publicación de la carpeta para otros usuarios no supone un riesgo alguno al entorno de seguridad.

[PAO CACHE v0.00]

Creation cache date : 10/05/2009

Creation token date : 17/42/47

[Aplicacion001]

user:99E1ACE257C81A27BB6BD72759A206CB54340C1304E6CEA3DFCCCA6BDF7FD9972C4B6E3BFD8D014A5D60F4648E6CE2E954 7414F334ACE12F71B3B157945A9623A03D817DA5D41BF130353172129ACF535369342BC76A2AAA694C0DFBF2EF23D42A0E38E0 63E66E946F0E4BF4A7C8912A0E56032F53D22A6BDD7AB861CBD5BA97

pass:353FF5F665AEF899A81E7A50F607E98C7C77E4F4409534248649D4C6923A519B885C8492D9844FC3D8283736E8127708F4A9 ECF32871B69F4A0BB476012A3230BB8857D118AB484394B277BE1CE6EA65127D54D12412C7C953FA79D05E4496427E1C9059277 794667CD4EDC9871F58AC4E7D90569CB26D5B9D0015DAE369803E

user:0CD70EF75C98579305F1559B9EB3C751BEC6DE8832B2186144758F08B958D6205AA0E11287E0767AC0881E39335A2226FEC 79577E0715A2DC8AA09B3A0F1250959E5FAF8975F65B4A07ECF970CB761B56AC634458AC7F1B99442791A71AA347278C7CB41D 83268CE236A31ED6A60A819A219ECE938ACF790DDA106B52418F879

pass:1056BDA87A212C54BDBF185B03D911C4B437834082E4EE426DC3AD2DEC0883306F2EAD2C59CE046CF61B91E76AA6BB403C 6CAB3032F8147258264CF994782490586C50317D753EE5AB00840B5CC595662EF4DA93972E82DF2E24C50D415C238F1A69C0A6B BA145C6A0BD396EF5F70642DE1C51185708D9FA2E44B1CD32D36E96

user:2DDAB973A2022D1E6D329A4978F1B37CBEFE850823FA15EE58741BE103AA982201A473A663333556452FC8773461579F3CD833CA43FEE2C567761392CF9932D9A778E52B6ACA3CBD435B11EC765EDFBBE38862C2D6F6B7DDDB80EF628A943C74DED2778083B42A51A7E645B4B344C6AAF73F632A297AE5350591FEAC8F33AF6B

pass:C27928B919E297E8C86960CF07A7BE761729A840B0BB86A767853AADB334AE24B9237543CB5A564D6E9912032ECBFBFFA7A1AC0034465356CF3A4990D13EF942D03DA92DE07F80B5F186262B96B37FC7FC3D73E8EBB158F88D76E91FBD1434E9AF003560779A543B619B4233D98F98AE11795067F4B3B64093E184326ABADE8A

[Aplicacion002]

user:42F8ECE1FC917AFA7E8A45A320044B45B6EC532978B6953D487F2342C37710DD2D71B49C9A3392ED48EAE2339ADCBCD0D0B72464431476F1D09E5A6C3F8F42737BB2CB7EE977619604C741584A1EB94AC1189BA0501BEBD585CEECF6CBA76706406EB559E852DFD110461B4BFD049435217CD8E0856D117C47614A02D29427C4

pass:914273198E73789934DCB48E39D60CFE846AD50DD23BCA05ED8C4F8BF29D74E5CB4943442D0FDCE60C6241DC2670FB76EF67022DDE824C5C7D70FE6AADDBD9FA6E4D762FF005C52B8EBA97DC9956859C4731F5CA073D4100DBFCBCB9F832F66C16F90F83837625589465C94C2C23BF3831FE520CE4713AF135160409B92D61FD

[Aplicacion003]

user:9D3FBA55B2EA19B50C5A6F792098D20B234D9242FF6EADC8B1204B0B82976141F4E39BCFBE5D42DA80C128743A61E78C4013001C4E44C4D1F52030DB9C5FE8B2F1936D1F12A41574290E0FBB707464E42435EF67A4B01F705FADC115168B70318AE62EEEE6DCC2AC2ECD54E18384C16E2B9D08C229D7974DAED45C1E84F82238

pass:1D03587AD624DF81DAA678E0501B3FB35F7CB34D5585580272954938EC91816CC12C4152D8F35C0E77D6E472880AE6901BB6EA6655FAF7A696C494CFC743B076D177BA9E51EC5407F08E8708F37D8885B3DD76DC054A99B0AA46606D9036FAFC1CD77E52FFAFAD4569500DFA4F1BD26F3525331258EC3442ABF61D93045EA2FC

user:D19160C669F128929F73803117DD0FE5EBAE12608F128F0EF3E1356052F109A1D64ACE13D8001C11019DB2A260346A5D784AC0EDAE995EA29731AB37F6A0178B20AF3A884B0ED1A9F6D2CC993F3171FAF214C049B7DB362FAC32171FB73998E8EDC420735C14AAD0AC0B8BBDEC2DA8086D21F538C332769E76683900C0B7A9F3

pass:C3FBDF4AC1A083DE559203DDFC3385FA41173DDC8C11FDC6909D294C87B52A57F1F995FDAA44157A2DDDD3D31BC682EDE8E87BB8D9BCBEE5B6E0C3F01D1F479C5E05519C19A050759B8CAAE7B47BEA2EB1C555EB3FE75C635CD133E4BD53015204ADDB6219B7D083D8057EA3FEB38291EFECD79FF4759AE6EF02E80663BD3509

4.2 Prueba de la aplicación PAOWS en todos sus parámetros

En este punto se va a proceder a probar la aplicación PAOWS con todos sus parámetros. Para testear cada uno de esos parámetros se analizará el resultado de cada una de las ejecuciones viendo en cada caso si el mensaje que aparece por pantalla concuerda con el esperado o, en caso de producirse una excepción, si esta excepción es esperada.

Comentar que durante estas pruebas se ha elegido un cifrado RSA-1024, cambiando la configuración por defecto de servidor.

4.2.1 create

En el siguiente punto se va a probar la creación del fichero *setup.ini*. La siguiente pantalla de consola muestra las diferentes opciones de ejecución cuando se utiliza la opción *create*.

Primero se ejecuta *create* sobre un directorio que no contiene ningún fichero *setup.ini*. Al ejecutar el comando *dir* se muestran todos los ficheros que contiene un directorio, se crea el fichero y se imprime un mensaje confirmando el éxito de la operación. Al ejecutar el mismo comando en un directorio que ya contiene un fichero *setup.ini* se deniega la operación indicando que debe incluirse el parámetro *force* para sobrescribir el fichero actual. Al ejecutar

con la opción *force* se muestra un mensaje de aviso que indica al usuario la criticidad de la operación y pidiendo confirmación, en caso de continuar el fichero *setup.ini* se sobrescribe.

Figura 40 Salida: Operación create

El fichero *setup.ini* contiene el texto mostrado a continuación de este párrafo. Subrayado en primer lugar se muestra el vector de inicialización automáticamente generado para un cifrado AES-256. En negro, entre las zonas de texto subrayadas, el par de claves RSA en formato XML cifradas AES-256 utilizando el vector de inicialización comentado anteriormente y la clave obtenida mediante hash SHA-256 a partir de la contraseña indicada por el parámetro *–pwd* al ejecutar la aplicación *paows*. En la última parte de texto subrayada se muestra la clave especificada por el parámetro *–pwd* cifrada RSA. Este último texto en verde se utiliza para comprobar el parámetro que se recibe cuando se ejecuta el comando *start|init*, como se verá posteriormente.

3B103AA37183209A1AF55413B14D0224:BCF41B7B53AE418972AE57321865EC1F5A8D71225B488D8498E996A6AD95C38E27A6E
3535B42342847C872C33B28FB7CDFDDE2E1087FECB889C63CD19849B655E843C5F03257C4F268F39DDA6148C2CEEDB7EB53512
BDEE745010DECFB9249417F8668F048E3F54BDDA1F6E892097DD67E2ACF834DA871CC0F33611915F317673DD38E6F54A613A05
144B2A87F9A14DB556DFD671928B6E9904B28EE66F837899DB1EBC79CA9D4A6082BDD966A3586D271D0477664613660538B6
31588E19F2F3C85075521F4522E0EA3DA4080FFF014543C39151B1F1F39976133AA8F0D1BA42EEED932688189B842D1540249816
4C1833F3FBA305B6754D7C9BF61B92479DBACD20B8E7E754951277731572E02E5FEE893AF2E3BAF08ECB6B4C2E22D9EA48287D6
EA0B875F5C52EF5C9D415D6CEB62F113BA3183FAC93DC05AAD72EA58BDCD176CEB892A880D71ABD18F1771E99FE352F0DFF228
679B9E2280E865968FA7EB5C63926FD9DF28D99949FBF90020E611DC4ACCC844ECBEED841CAB4A2C56458770BCC90370DF9C3C
17507F408912E58822744C7733CBB972D4AFE8DDEF16AB9A90B005B05D4A5E77FCE74DBA0D5AAE1FA855C538745DCA958A1DE
50B370D7D7A6C0B91523B2BD3E14C3FB92EB601B09E11F391C983AFF9A56C869388B40AC4BFFD719EDF2410456C188057BE99A
F1D9355FA7D26ABA199C7AD83397C6050D1D07B442B4649B4D41FB94164277248A12263D17FC203A057D19B79BBE55D8A1830
40E8218A8AF3ED4C4D1103F735BFD285C24ACDB20BBF063A690E0AC6977835941C01FC5B6222D42018324ADB5837E4F0DF1984
49F2ACDEDC2010BD121130B0B844F0F61D9535DC6A13C35F1FA9A17A6269613338BC35D26246D2199FC0FC9934397D54EFE622
0B027EA05066643122C9F1085F81276D1CF8507ABC2A6A6645D92F16B075A6FD80350F7EAD45B5112C51C03B12769773438EEF9
99B34DA9C2E8519BEDE65514D1991FF45BCE801D8C2DE09B47C7DE1DC83C480A3A3EE23F0210E992DFB77EA508853C51C9B8B2
0449667BCDF85CEE2F1BF0192668C43DF33EA6A4FD4E6A24BBBE3CC927AB9360F58AF56DE2A04CF2BC2346ACB368A8A72AA2E5
9E2B1F9773F172BC6D07CF3787FDF63E6C0761C176F47F59F32C754C8129C8C9D956DA8103CC4AA4247D19B95F935E371809EED
921FA28F7FE007ACC17943E25F80B52BB10D0D0CECE5599F594BE538608FD91453D221BAAF659B16462415797FA7034A34B53D3
714C5F2B8E5649B63A202E21E7BADE1C6DAF544C32748DF94C:851FEAA5FD8088F25AE7F7586F9271909446F7626D849EB645A4
2CC8E0759B2891ADC8C7980F86748F4494D610CCE1464066688D3B3BF08C4290AED5B8F8544767BD3BC5F0223678EB5488806C5
D41514D05976B2C8700B5ED5882533B9C9EB18437930D03E8E0295047C476234DF03063C7385267451BABE6D84DF702A142EC

4.2.2 start|init

En este apartado se probarán los diferentes escenarios para el comando *start* o *init* que se utilizan indistintamente. Ambos comandos arrancan el proceso en modo desasistido *PAOServer.exe* que debe encontrarse en el mismo directorio donde se ejecuta el comando con la aplicación *paows*. Este comando comprueba dicha situación, así como la posibilidad de que exista otro proceso *PAOServer.exe* corriendo aunque sea en un puerto diferente ya que en el mismo puerto daría un error.

La siguiente prueba muestra ambas situaciones así como el arranque del servidor con una contraseña incorrecta y el arranque del servidor de la forma apropiada. Nótese que si se ha intentando arrancar el servidor con la contraseña inapropiada primero debe cerrarse para proceder con el proceso de inicialización desde el principio. Esto puede parecer un tanto ineficiente, pero la realización de esta manera es un criterio de diseño impuesto por el cliente.

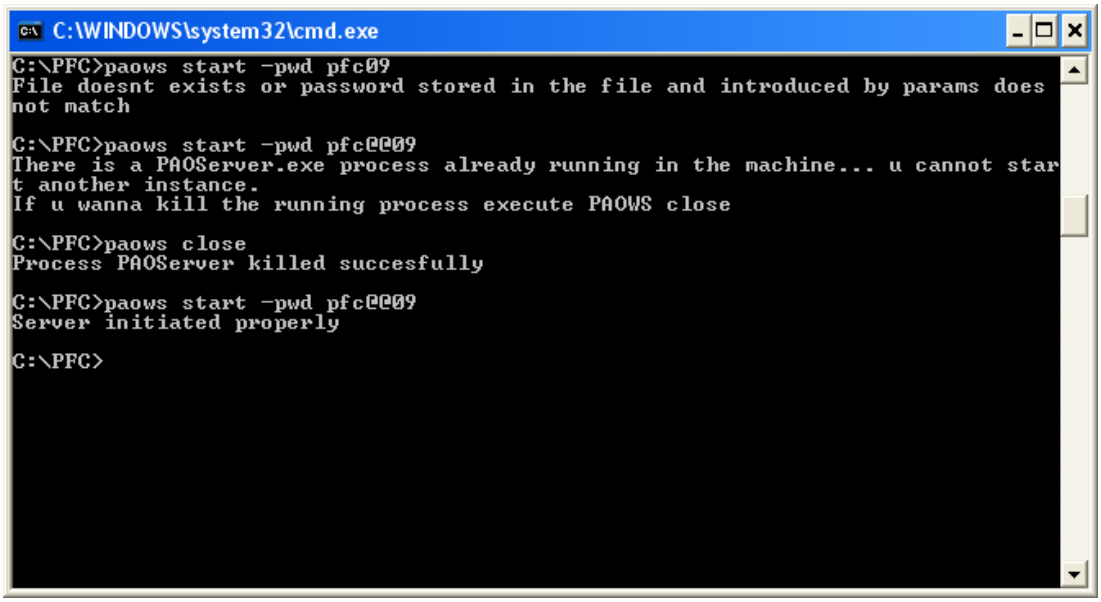


Figura 41 Salida: Operación start

4.2.3 close|stop

La funcionalidad realizada por *close* o *stop* es la más sencilla de todas. Simplemente busca entre todos los procesos en ejecución dentro de la máquina local y si encuentra alguno con el nombre *"PAOServer.exe"* lo elimina del sistema.

No requiere de contraseña por parte del usuario, ya que el servidor únicamente puede ser accesible desde el propio servidor donde se ejecute el proceso PAOServer. Puede pensarse entonces por qué hace falta contraseña para la inicialización del servidor. Este caso es completamente diferente porque la inicialización requiere la lectura de la contraseña que guarda todas las contraseñas de los usuarios. Se entiende, dentro del cliente, que cualquier administrador es responsable para ejecutar un comando del que desconoce su impacto en el sistema, pero no todos los administradores con acceso a la máquina tienen que conocer la contraseña que les da acceso ilimitado al sistema.

Las siguientes figuras muestran el administrador de tareas de Microsoft en el que puede observase como el proceso desaparece después de la ejecución del comando.

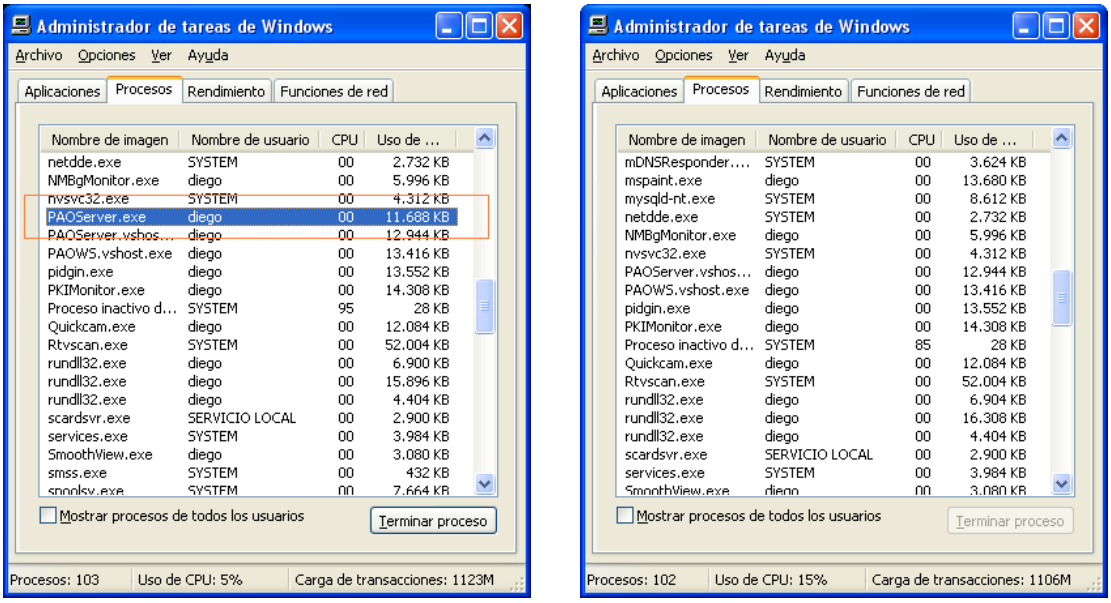


Figura 42 Administrador de tareas

NOTA: Los procesos con extensión .vshost.exe son procesos imagen del Visual Studio que permiten la depuración por ello en la figura de la derecha aparece un proceso con nombre PAOServer.vshost.exe que fijándose bien está también en la figura de la izquierda y, por tanto, no sólo no debe ser eliminado sino que en caso de eliminarse sería un error de la aplicación.

4.2.4 backup, gencode y getpwd

La función *backup* de la aplicación realiza la copia de respaldo de la clave simétrica AES de un usuario. La ejecución del proceso no tiene ninguna salida salvo que el servidor TCP/IP no haya sido inicializado de manera correcta. En caso de que la operación de copia de respaldo se realice con éxito este comando no ofrece ninguna salida. Como se ha comentado anteriormente, este comando busca el usuario que se especifica con la opción *–u* en la estructura de datos que se tiene en memoria y en caso de encontrarle le cambia la contraseña

por la especificada con la opción –key, o bien, añade la información pasada por parámetro a la comentada estructura de datos. En cualquiera de los casos, esto produce una re-sincronización de la estructura de datos contenida en memoria al fichero contenido en la memoria física del ordenador.

La función *gencode* genera un código aleatorio que permite a un usuario recuperar su contraseña. Al igual que cualquier otra opción del programa requiere que el servidor TCP/IP haya sido inicializado previamente. En caso de que el usuario especificado en los parámetros no esté en la estructura de datos del servidor se muestra un error indicando dicha situación. Si el usuario existe el servidor guarda una entrada con el siguiente formato:

hora_actual_del_servidor;valor_sha_del_codigo_generado

La función *getpwd* obtiene la contraseña simétrica de un usuario para lo cual hay que especificar el código aleatorio que se ha generado desde el CAU (indicado en la opción –g) y el usuario del que se quiere recuperar la contraseña (especificado con la opción –u), siempre y cuando la ejecución de este comando se produzca en el número de minutos configurable, 30 por defecto. Para que se produzca comparación el usuario debe existir en la estructura de datos del servidor y debe haber solicitado un código para la recuperación de su contraseña, ya que en caso de que alguna de las dos condiciones no se cumpla el sistema produce un error directamente, sin necesidad de hacer ninguna comparación.

El mejor escenario de prueba es, por tanto, realizar la copia de respaldo de una contraseña de un usuario y luego tratar de recuperar dicha contraseña y ver que ambas coinciden. Durante el proceso se entiende que el servidor TCP/IP está correctamente inicializado, ya que los errores producidos por este hecho se han analizado con detenimiento en los apartados anteriores.

Para probar los posibles errores se han definido tres escenarios diferentes. En el primero de ellos se ha especificado un único minuto de demora para la recuperación de la contraseña desde que se ha generado el código de autenticación de manera que se pueda observar el error por demora en la ejecución. En la segunda de ellas trata de generar un código para un usuario que no existe y, la tercera, trata de recuperar una contraseña por medio de un código que, simplemente, no se corresponde con el código del usuario.

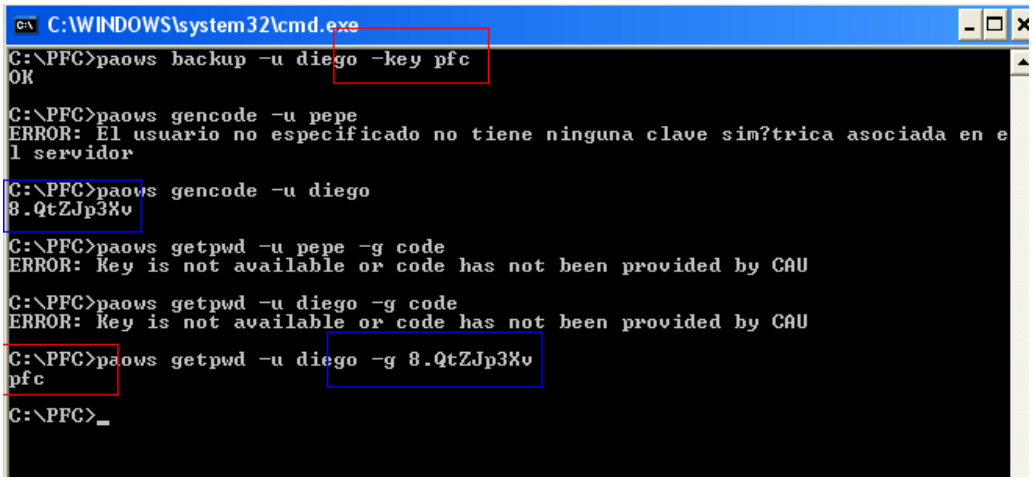


Figura 43 Salida: Operaciones backup, gencode y getpwd

En la ejecución puede observarse en los cuadros primero y cuarto como la contraseña que se ha guardado para el usuario mediante el proceso de copia de respaldo es el mismo que se obtiene como resultado mediante el proceso de obtención de la contraseña. Y en los cuadros segundo y tercero puede observarse como el código generado debe ser el mismo que el código que la aplicación haya generado. La siguiente ejecución muestra el resultado de un retraso excesivo cuando el servidor ha sido configurado para no aceptar retrasos mayores de un minuto, así mismo muestra como se puede sobrescribir la contraseña de un usuario.

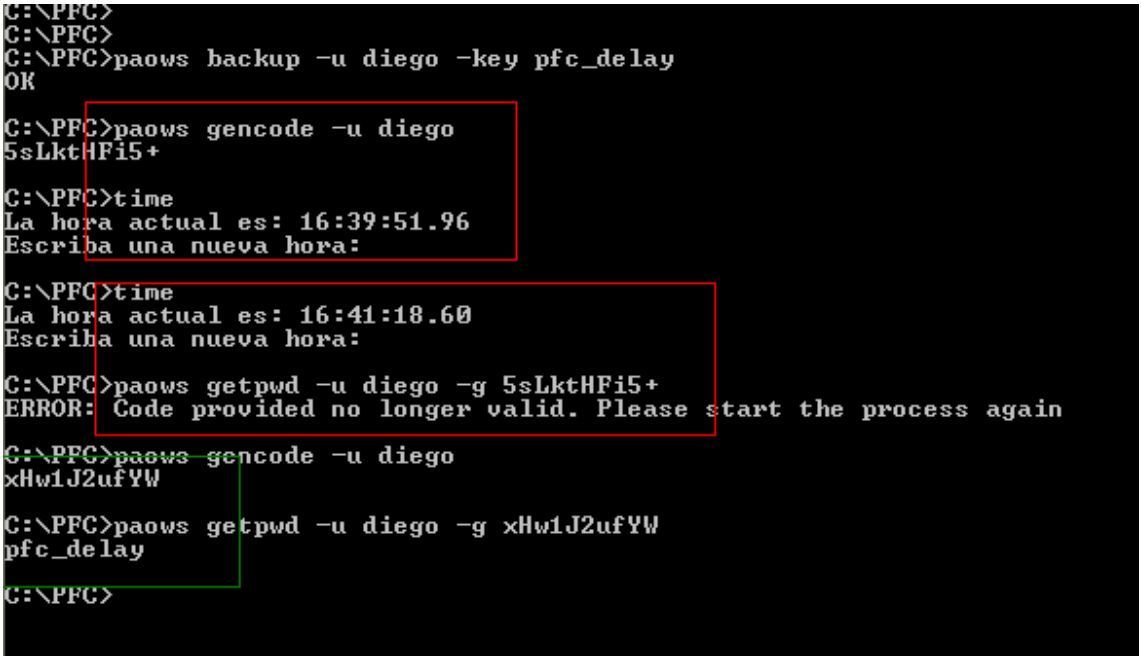


Figura 44 Salida: Error en la obtención de la contraseña

4.2.5 changemasterpwd

Esta opción cambia la contraseña almacenada en el fichero *setup.ini* mediante la cual se arranca el servidor de claves TCP/IP, por ello, cuando se especifica este parámetro el servidor TCP/IP debe cerrarse, ejecutando un proceso similar al que se ejecuta cuando se invoca esta misma aplicación con el parámetro *close*.

Debido a este motivo la aplicación muestra un mensaje de aviso indicando la situación y ofreciendo al usuario la posibilidad de cancelar el proceso. Si el usuario decide no cancelar y no hay ningún error, las contraseñas se cambian. El único error que puede existir es que la contraseña antigua no sea la correcta. El caso de desconexión del servidor durante el proceso que ocurre es el mismo al descrito en el apartado 4.5.2. En caso de que no exista error el proceso del servidor TCP/IP de seguridad se arranca con la nueva contraseña.

Para demostrar el correcto funcionamiento de la aplicación se va a provocar un cambio de contraseña y acto seguido se intentará levantar el servidor TCP/IP para ver el resultado.

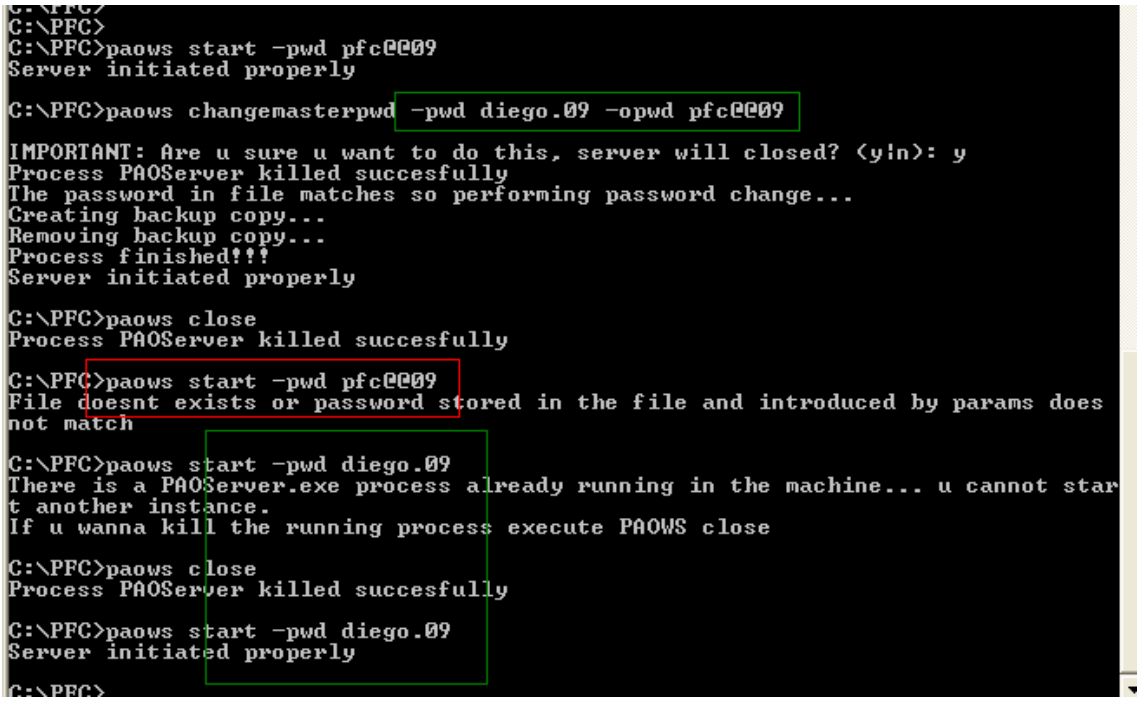


Figura 45 Salida: Operación cambiar la contraseña maestra

En el segundo cuadro se muestra el correcto funcionamiento de la aplicación y cómo, una vez realizado el cambio de contraseña, ésta no puede utilizarse para arrancar el servidor de claves TCP/IP.

4.3 Integración de la API con el servicio WEB

En el proceso de integración de la API con el servicio web se ha de comprobar el correcto funcionamiento de las dos operaciones web que realiza el servicio *pao.asmx*. Para ello se van a realizar pruebas de dos maneras posibles, una de ellas utilizando el navegador y, otra de ellas consiste en la clonación del dispositivo, lo que requiere que el primero de los dispositivos haya realizado una copia de respaldo del certificado y que el segundo de los dispositivos obtenga esa contraseña que se guardó en el servidor de claves.

El servicio web no es más que una pasarela para que los usuarios sin acceso físico al servidor puedan ejecutar las operaciones de copia de respaldo y de obtención de la contraseña desde fuera del mismo.

4.3.1 Ejecución mediante el navegador

La principal misión de un servicio web es proporcionar un entorno de ejecución distribuido de manera que las capacidades del servidor estén accesibles desde cualquier punto. Así, por ejemplo si se pone la URL donde el servicio web está en pre-producción, podemos obtener un listado de todas las operaciones que puede realizar el servicio web. Esto puede observarse en la siguiente figura.

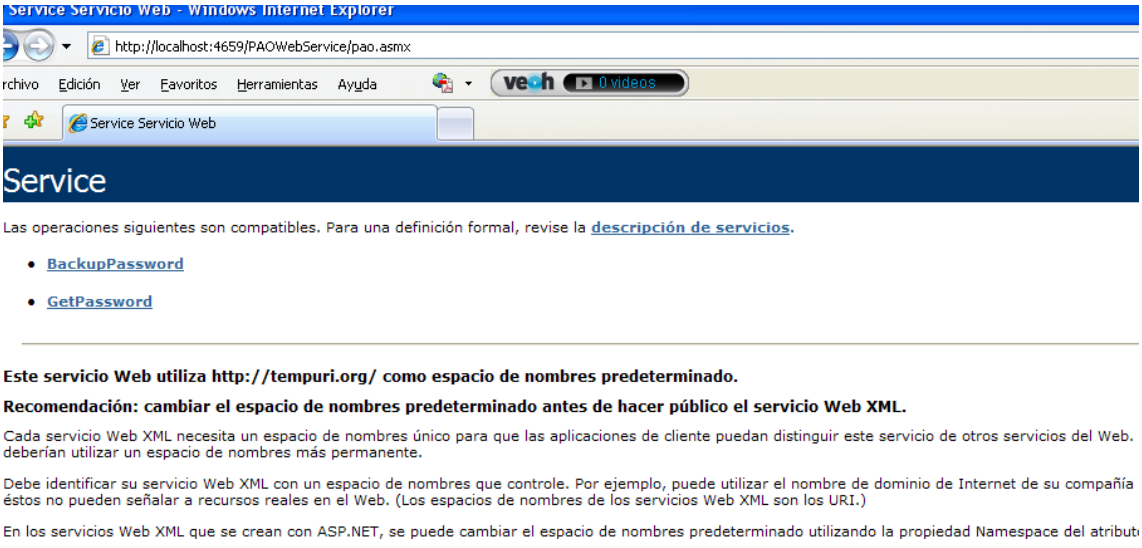


Figura 46 Navegador con el listado de operaciones del servicio web

Si se hace clic en cada una de las operaciones se obtiene una descripción del mensaje SOAP en las versiones 1.1 y 1.2 o el mensaje HTTP POST que debe enviarse al servidor para poder realizar cada una de las operaciones, ofreciendo también la posibilidad de ejecutar dicha operación directamente sobre el servidor observando el mensaje SOAP de respuesta. Así, por ejemplo, se observa el mensaje SOAP para la operación *GetPassword* en la siguiente figura.

Figura 47 Descripción SOAP de GetPassword

En la parte superior de la página se puede ejecutar dicho servicio directamente desde el navegador.

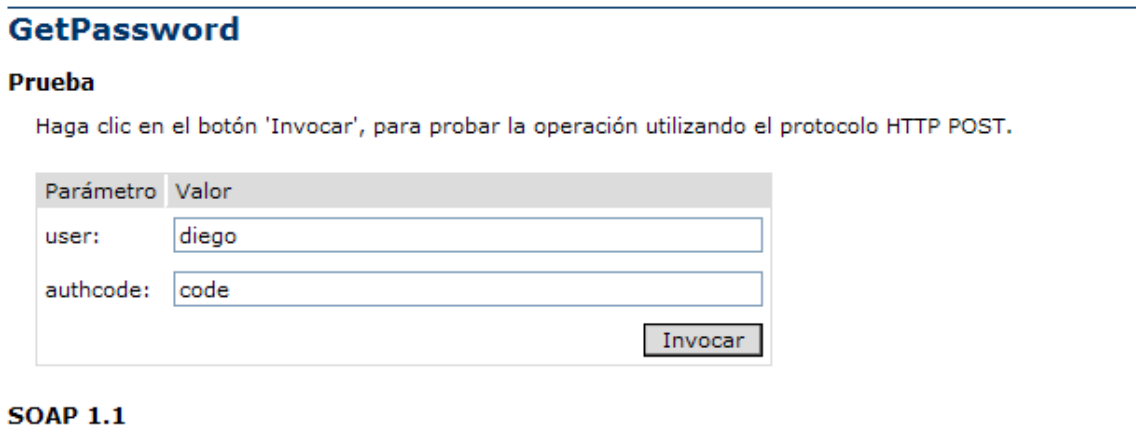


Figura 48 Ejecución del servicio desde el navegador

Obteniendo así el mensaje de respuesta de la siguiente figura.

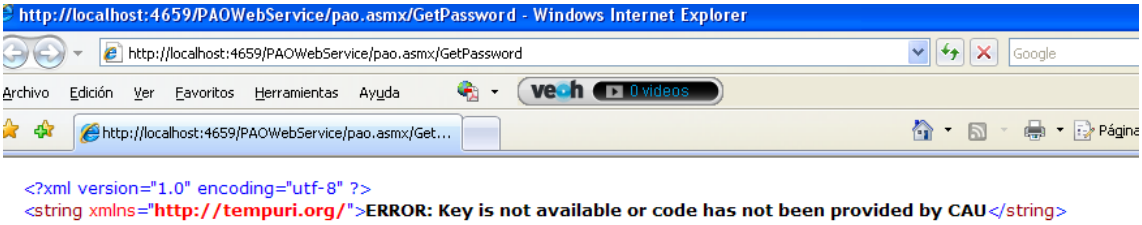


Figura 49 Respuesta del navegador

NOTA: El resultado de esta prueba es el esperado según las consideraciones y los escenarios descritos en el punto anterior.

4.3.2 Clonación del dispositivo de usuario

Para comprobar la funcionalidad de sendas operaciones y la integración de la librería con el servicio web se ha considerado como una prueba irrefutable la descrita en la introducción de este apartado por la que se realizará una clonación de dispositivo, simulando el proceso que ejecuta la compañía en caso de extravío o robo de uno de sus dispositivos. El código que se va a utilizar para la prueba es el siguiente:

```
Imports System.Diagnostics
Imports System.IO
Imports System.Text.RegularExpressions
Imports System.Security.Principal

Module Test
    Sub Main(ByVal Args() As String)
        Dim token As PAOAPI.PAOToken
        token = New PAOAPI.PAOToken()
        Dim filename As String = "pao.txt"

        If Args.Length <> 0 Then
            Dim pkcert As String = ""
            Dim tkDate As String = ""
            Dim outCert As String = ""
            Dim encodedPin As String = ""
            Dim login As String = ""
            Dim ErrorDesc As String = ""
            token.CreatePAOToken(pkcert, tkDate, outCert, encodedPin)
            Console.WriteLine("Creado token con fecha {0}", tkDate)
            Dim encoded = token.AsymmetricEncrypt("Mensaje de texto cifrado")
            Console.WriteLine("Mensaje En Claro {0}:", "Mensaje de texto cifrado")
```

```
            token.CloseToken()
            Console.WriteLine("Ejecutando el proceso de generacion de un codigo para el
usuario")
            Dim authcode As String = ExecProcess("PAOWS.exe", String.Format("gencode -u
{0}", WindowsIdentity.GetCurrent().Name))
            Dim filecontent As String = ""
            filecontent += "Encoded Pin: " + encodedPin + Environment.NewLine
            filecontent += "Out Cert: " + outCert + Environment.NewLine
            filecontent += "Date: " + tkDate + Environment.NewLine
            filecontent += "Auth Code: " + authcode
            filecontent += "Message: " + encoded + Environment.NewLine
            File.WriteAllText(filename, filecontent)
            Console.WriteLine("Fichero pao.txt creado")
        Else
            Console.WriteLine("Ejecutando el programa para crear un nuevo token...")
            Console.WriteLine(ExecProcess("PAOAPI_Test.exe", "Create"))
            Dim token2 As PAOAPI.PAOToken = New PAOAPI.PAOToken()
            Console.WriteLine()
            Console.WriteLine("A continuacion se muestra el contenido del fichero creado
en la ejecucion anterior...")
            Dim filecontent As String = File.ReadAllText(filename)
            Console.WriteLine(filecontent)
            Dim PIN As String = token.InitUserPIN()
            Dim Pattern As String = "Encoded Pin: (?<encodedpin>.*$)\n" + _
                                    "Out Cert: (?<outcert>.*$)\n" + _
                                    "Date: (?<tkdate>.*$)\n" + _
                                    "Auth Code: (?<authcode>.*$)\n" + _
                                    "Message: (?<message>.*$)\n"
            Dim m As Match = Regex.Match(filecontent, Pattern, RegexOptions.Multiline)
            Console.WriteLine("Ejecutando la recuperación del token...")
            token2.RecoverPAOToken(PIN, RemoveLast(m.Groups("authcode").ToString),
RemoveLast(m.Groups("encodedpin").ToString), RemoveLast(m.Groups("outcert").ToString),
RemoveLast(m.Groups("tkdate").ToString))
            Console.WriteLine("Contenido del mensaje descifrado...")

Console.WriteLine(token2.AsymmetricDecrypt(RemoveLast(m.Groups("message").ToString)))
            Console.Read()
        End If
    End Sub

    Function ExecProcess(ByVal Filename As String, ByVal Args As String) As String
        Dim p As Process
        p = New Process()
        p.StartInfo.FileName = Filename
        p.StartInfo.WindowStyle = System.Diagnostics.ProcessWindowStyle.Hidden
        p.StartInfo.Arguments = Args
        p.StartInfo.UseShellExecute = False
        p.StartInfo.RedirectStandardOutput = True
        p.StartInfo.StandardOutputEncoding = System.Text.Encoding.ASCII
        p.Start()
        Dim output As String = p.StandardOutput.ReadToEnd()
        p.WaitForExit()
        Return output
    End Function

    Function RemoveLast(ByVal str As String) As String
        Return str.Remove(str.Length - 1)
    End Function
End Module
```

Todos los extractos de código mostrados hasta el momento son autoexplicativos o muestran el cómo invocar algunas librerías de la API para su correcta utilización dentro del entorno. Sin embargo en este caso el código es algo más complejo por lo que se quieren destacar los siguientes puntos:

- El código debe ejecutarse sin argumentos, esto provoca que se ejecute el *Else*, es decir, que se arranque otra instancia de este programa y se ejecute con argumentos.
- La ejecución con argumentos guarda en un fichero de texto plano los valores cifrados del PIN, del certificado y la fecha de creación del dispositivo de almacenamiento

seguro, así como un mensaje cifrado que demuestra que la clonación del dispositivo es total.

- El fichero de texto simula el servidor LDAP de la red, ya que todos estos valores deben guardarse y obtenerse mediante peticiones LDAP seguras.
- Los valores del fichero de texto se leen mediante expresiones regulares y a la finalización del proceso de usuario se muestra el mensaje de texto fruto del descifrado con el nuevo dispositivo clonado.

La ejecución del código no es lineal y requiere de dos procesos para su correcta finalización. Esto es debido a que el código C++ no administrado proporcionado por Aladdin no permite la inicialización del PIN más de una vez por cada ejecución de su librería. Este hecho se comentó y trató de solventar con Aladdin sin acierto ya que la conclusión final fue que sucesivas librerías de su driver sí permitirían dicha circunstancia, pero no se proporcionó ningún parche para la aplicación actual. De igual modo, la aplicación no permite el cambio de dispositivo durante el mismo proceso. Esto se debe a que el .NET Framework asigna un identificador interno a cada dispositivo de hardware dentro de la aplicación y el aviso de dicha sustitución al Framework conllevaba una alta complejidad en código para ser un *workaround* similar al que se realiza ejecutando la aplicación en procesos diferentes.

La salida del programa puede observase en las dos siguientes figuras. El contenido del mensaje de texto en claro que se quiere cifrar es el mismo que el contenido descifrado del último recuadro en la siguiente figura.

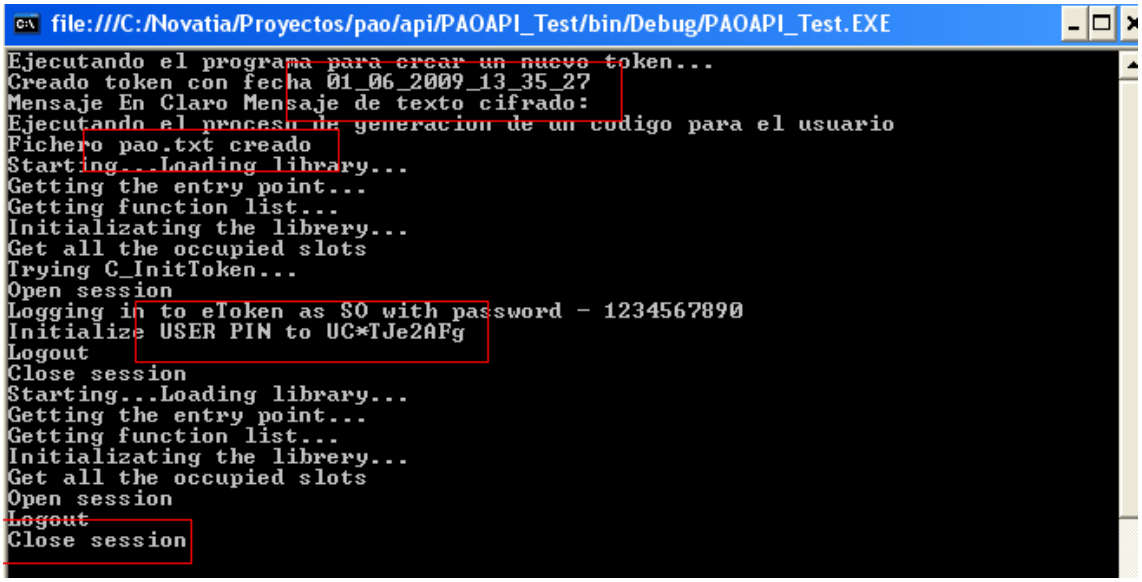


Figura 50 Salida: Clonación del token I

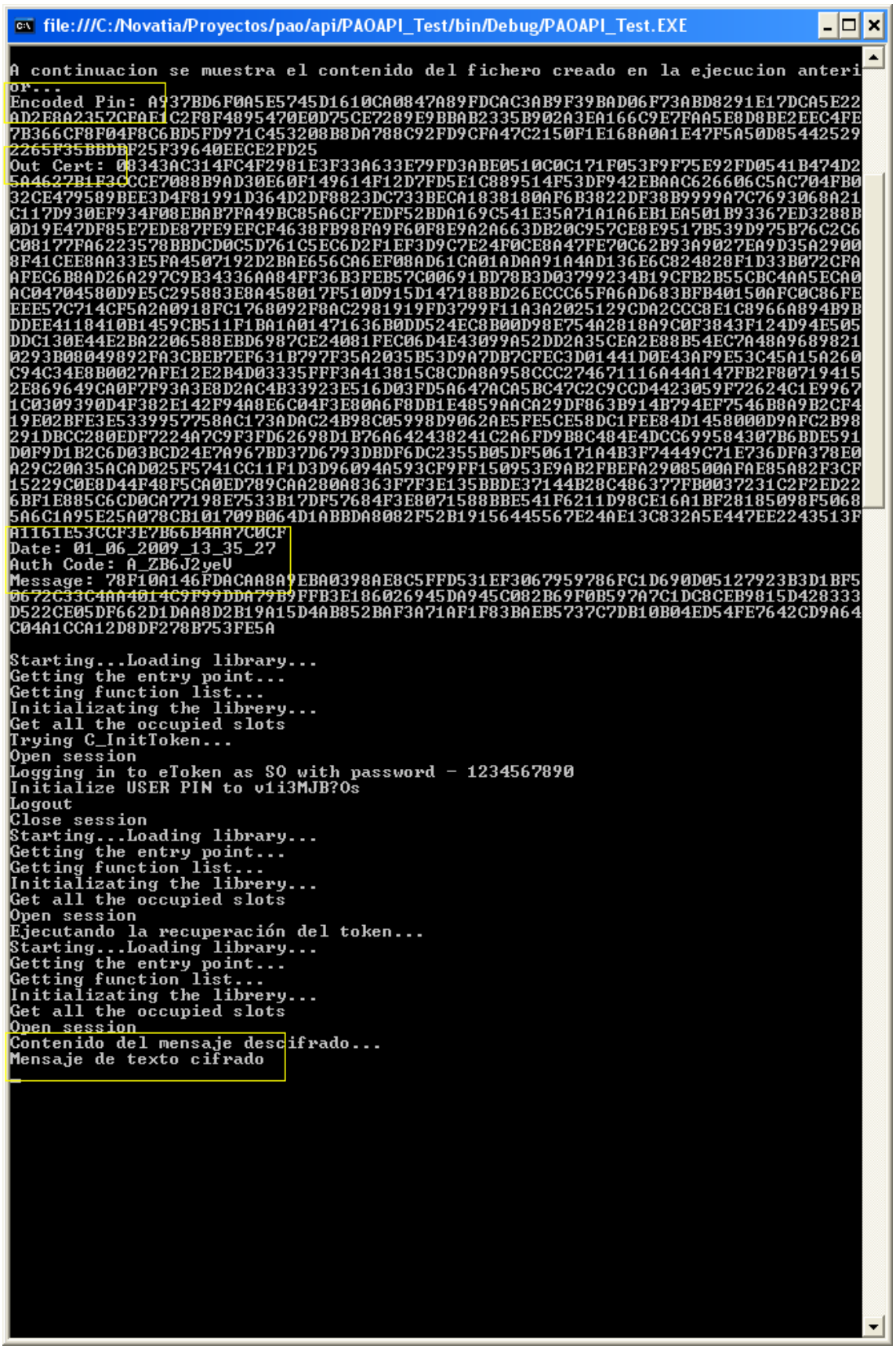


Figura 51 Salida: Clonación del token II

4.4 Escenarios de seguridad

4.4.1 Seguridad en los servicios web

Cuando por vez primera se intentó realizar esta prueba de seguridad se determinó que dado los parámetros de entrada de los servicios web, cualquier usuario con conocimientos básicos de programación podía:

- Realizar una copia de su clave de seguridad no autorizada en su nombre.
- Realizar una copia de su clave de seguridad no autorizada en nombre de otra persona, creándola si ésta no existiese o sobrescribiéndola (caso más grave) en caso de que sí que existiese.

Sin embargo la arquitectura no permitía recuperar una clave de seguridad que no fuese la propia ya que requiere del código de autenticación. Obviamente este escenario presenta una brecha importante en la seguridad que debía ser subsanado. Para esto lo mejor es el uso de la autenticación integrada en el dominio.

Mediante este tipo de seguridad el nombre de usuario se escribe en el contexto de la sesión que consume los servicios de manera que el propio servidor es capaz de validar qué usuario está haciendo uso de los servicios publicados y si éste ha sido validado en el dominio o no.

Para ello se han realizado las siguientes modificaciones en los servicios web, añadiendo el extracto de código que se muestra a continuación:

```
if (!Context.User.Identity.IsAuthenticated)
        return "ERROR: User not authenticated in domain";
user = Context.User.Identity.Name;
```

De esta manera únicamente los usuarios autenticados en el dominio pueden hacer uso de los servicios web publicados dentro de la arquitectura e independientemente del usuario al que quieran suplantar, sólo podrán usar los servicios bajo su propio nombre.

Esto nos lleva a las siguientes consideraciones:

- Los usuarios siguen pudiendo consumir los servicios web de manera no autorizada. Sin embargo únicamente provocarían un impacto sobre el acceso a sus propios servicios. Si bien esta situación no es idílica se puede corregir utilizando la siguiente solución:
    - El servicio web acepta un parámetro más; una clave escrita en código. Como bien es sabido la seguridad por oscuridad no es nunca una solución plena y cualquier usuario con capacidades o conocimientos de descifrado podría interceptar sus propios paquetes descifrar la clave SSL y obtener dicho secreto compartido.
- Los administradores no pueden crear dispositivos seguros para otros usuarios, ya que el dispositivo se asocia al usuario validado en la sesión de Windows activa en ese momento. Aquí se plantean dos soluciones alternativas:
    - Utilizar diferentes servicios para los administradores y para los usuarios normales del dominio, de manera que la seguridad integrada comprobase también si el usuario autenticado es miembro de algún grupo con altos privilegios.

- Que los administradores utilicen la consola para crear las claves en el servidor de seguridad, haciendo uso del programa PAOWS.exe descrito en apartados anteriores. Dicha aplicación de consola no comprueba la integridad de los usuarios ya que sólo está accesible en el servidor donde los administradores son los únicos roles del sistema con acceso.

Estas soluciones alternativas no han sido implantadas en la solución final ya que el cliente decidió que si un usuario quería o de alguna manera intencionada sobrescribiría sus propias contraseñas se le negaría el acceso a las aplicaciones. En cualquier caso una inclusión de un registro de operaciones contra un fichero accesible exclusivamente por administradores puede aclarar los diferentes casos que se producen con intención de los usuarios.

4.5 Tolerancia frente a fallos

4.5.1 Desconexión del servidor mientras realiza un volcado de los datos de memoria a los datos en disco

Los datos son volcados de memoria a disco siempre que se produce un cambio en la colección de objetos accesible desde la memoria de la máquina.

Durante este proceso si se produjese un apagón en el servidor se podría corromper el fichero de claves. Siendo conscientes de esta situación los datos son grabados a un fichero temporal de manera que si se produce un error en un punto intermedio es el fichero temporal el que queda interrumpido y el que sufre las consecuencias. Cuando todos los datos son guardados en el fichero temporal se borra el fichero original y se renombra el temporal para que coincida con el original. De esta manera el apagón puede producirse en tres puntos:

1. Cuando el fichero temporal no está completo totalmente. En este caso el fichero original contiene todas las claves menos el último cambio que se ha producido en el servidor. Se pierde, pues, únicamente el cambio más reciente.
2. Antes de borrar el fichero original. Aunque el administrador podría borrar el fichero original y renombrar el temporal no es una práctica recomendada ya que nunca se puede tener la absoluta certeza del porcentaje de usuarios válidos que quedan en el fichero temporal. Por tanto, se recomienda borrar el fichero temporal y rearrancar el servidor, perdiendo así el cambio más reciente que se haya producido en la plataforma.
3. Cuando el fichero original se ha eliminado y antes de que el temporal haya sido renombrado. En este caso al arrancar el servidor TCP/IP de seguridad se produce un error porque no encuentra el fichero *users.pkz* y el administrador no tiene más que renombrar manualmente dicho fichero y rearrancar el servidor para que los cambios estén operativos en el sistema.

4.5.2 Desconexión del servidor mientras se produce un cambio de contraseña simétrica

Al igual que el caso anterior cuando se produce un cambio de contraseña simétrica se realiza sobre un fichero temporal para luego realizar los cambios. En este caso no hay pérdida de datos y el peor escenario es cuando la contraseña simétrica queda sin cambiar por más del

plazo máximo permitido en la arquitectura. Al igual que el caso anterior, se definen 3 escenarios diferentes en los que el proceso puede ser interrumpido:

1. Cuando no se ha completado el fichero temporal. En este caso se elimina el fichero temporal y la clave simétrica se mantiene durante otro período más.
2. Cuando el fichero temporal se ha completado, pero el original no se ha borrado. Al igual que el caso anterior, se recomienda que el administrador borre el fichero temporal y prolongue el cambio de contraseña durante otro período más, ya que al igual que el caso 2 del apartado anterior no se puede saber con exactitud el porcentaje de usuarios cifrados con la nueva contraseña que se han guardado en el fichero temporal.
3. Cuando únicamente tenemos el fichero temporal, el administrador no tiene más que renombrar manualmente el fichero. En este caso no se produce pérdida de información alguna.

4.5.3 Servidor arrancado, pero no inicializado correctamente

Este escenario se ha producido y comentado durante las pruebas de la aplicación *PAOWS*. En este caso es necesario cerrar el proceso y tratar de arrancarlo de nuevo con la contraseña correcta.

5 Conclusiones y trabajos futuros

5.1 Conclusiones

La arquitectura que se ha desarrollado es plenamente funcional y actualmente da servicio a cientos de usuarios como plataforma común de acceso a sus aplicaciones corporativas.

Mediante esta arquitectura los usuarios son capaces de acceder a todos los pares usuario-contraseña de todas las aplicaciones corporativas de manera segura, almacenarlos en una caché en el ordenador local y utilizarlos para el acceso a las mismas.

Por otra parte, se ha desarrollado un servidor TCP/IP seguro expuesto mediante servicios web a todos los usuarios de la red de manera que puedan hacer copias de respaldo y recuperar sus claves simétricas cuando se considere oportuno. Dichas claves simétricas se utilizan para recuperar o clonar los dispositivos extraviados o robados sin que dicho suceso implique una carga extra de trabajo para el departamento de IT.

Cada usuario dentro del sistema es poseedor de un par de claves RSA contenidas en un dispositivo de almacenamiento seguro y una clave simétrica que le permite recuperar dicho par de claves cuando sea necesario.

El proyecto ha permitido el desarrollo de una driver en C++ para interactuar utilizando la librería estándar de PKCS#11 con un dispositivo de almacenamiento seguro. Se ha desarrollado una API para que cualquier aplicación gráfica pueda, de la manera que considere oportuna, hacer uso de esta infraestructura y se han tenido que definir procesos generales para que esto sea realmente así y no únicamente un propósito de diseño.

Mediante el desarrollo de un servidor TCP/IP y una aplicación cliente, así como mediante la creación de un servicio web simple se ha completado un ciclo por la tecnología .NET que abarca desde el bajo nivel del driver a escenarios de arquitecturas distribuidas propias de entornos mucho mayores. Además el uso de las bibliotecas criptográficas ha permitido la asimilación de los protocolos, así como sus particularidades y conocer y saber identificar, al mismo tiempo, en qué contexto y qué usos tienen cada uno de ellos en el mercado real actual.

5.2 Trabajos futuros

Sin embargo la arquitectura, como casi siempre, es mejorable en algunos aspectos. Todas las ideas se identificaron para realizar una nueva versión y se reflejan en los siguientes puntos:

1. Se considera ineficiente que los usuarios tengan que disponer de un dispositivo de almacenamiento seguro para el acceso a la plataforma ya que supone un gasto elevado a la hora de ofrecer dicho servicio a todos los usuarios de una empresa grande. Por ello se plantea una modificación de los servicios web y del aplicativo para que detecte si el usuario tiene o no insertado un dispositivo seguro y en caso de no tenerlo que realice peticiones al servidor TCP/IP mediante los servicios web para obtener un certificado asignado que le permita acceder a las funcionalidades SSO de la arquitectura.
2. Se considera ineficiente que los usuarios sean guardados en un fichero de texto plano ya que el acceso a disco es mucho menor. Por ello futuras versiones sustituirán dicho

fichero de texto por un servidor de base de datos como MySQL o SQL Server Express Edition ambas versiones gratuitas de bases de datos disponibles para los usuarios de Internet.

3. Los servicios web lanzan la aplicación PAOWS por lo que dependen de éste proceso para satisfacer la petición. Por ello, si existiese una base de datos sería mejor realizar un servicio de Windows que atendiese dichas peticiones mejorando el acceso a la base de datos.
4. La inclusión del par de claves RSA dentro de los usuarios que posean el dispositivo limita la funcionalidad de éste. Dicho dispositivo aunque utilizado dentro de la red corporativa únicamente para el acceso a la arquitectura posee muchas otras funcionalidades que pueden aprovecharse. Entre otros se citan:
    a. Acceso automático al sistema operativo.
    b. Almacenamiento de documentos privados.
    c. Acceso a los usuarios y las claves de usuario almacenadas en los navegadores de Internet del sistema operativo.

    Para ello se requiere que dicho dispositivo tenga almacenado un certificado de usuario, por ejemplo X.509, en vez de un par de claves.

5. Si el dispositivo almacena un certificado en vez de un par de claves, dicho certificado puede ser:
    a. Autogenerado y no validado con ninguna CA y, aunque permite las funcionalidades descritas en el punto anterior limita otras como el acceso a las plataformas públicas mediante el uso de certificados reconocidos.
    b. Firmado por una CA no reconocida. En caso de que el cliente obtenga un acuerdo con una CA externa o decida montar su propia infraestructura de clave pública, esto dota a los usuarios de una mayor seguridad así como la posibilidad de acceso a un amplio abanico de servicios ofrecidos por las plataformas o por el propio dominio corporativo.
    c. Certificado reconocido, firmado por una CA propia con un acuerdo con una empresa de terceros reconocida o directamente por una CA reconocida en caso de que el cliente final no decida instaurar una infraestructura propia. En este caso los usuarios tienen acceso tanto a todos los servicios que la propia red corporativa ofrezca así como a todos los servicios que Internet ofrezca a todos los usuarios con certificados reconocidos: declaración de la renta,…
6. La migración a Linux es un aspecto fundamente que deberán abordar los futuros desarrollos. Si bien el uso de servicios web presupone la interoperabilidad de los sistemas operativos, el primer paso debería ser realizar una versión de la librería PAOAPI.dll para el entorno Linux de tal forma que la infraestructura incluyese también a los usuarios de dicho entorno.

Así por tanto, aunque como desarrollo planteamiento inicial la arquitectura propuesta es una buena base de desarrollo para los servicios de criptografía y de seguridad de la red corporativa. Aunque siempre se pueden dotar de muchos otros ampliar los servicios que para facilitan la vida tareas de los usuarios tanto dentro como fuera de la red de la empresa, mejorando así tanto la experiencia como la satisfacción del usuario.

Bibliografía y referencias

Las referencias aquí detalladas se presentan en el orden de citación del texto. Todas las URL fueron accedidas por última vez durante el mes de Septiembre de 2009 para comprobar su correcto funcionamiento.

1. **Microsoft Corporation.** .NET Framework Conceptual Overview. [En línea] 2009. http://msdn.microsoft.com/en-us/library/zw4w595w(VS.80).aspx.

2. **Jones, Trevor.** Grounding Blogs. [En línea] 2008. http://grounding.co.za/blogs/trevor/WindowsLiveWriter/Overviewofthe.NETPlatform_BD9D/image_thumb_1.png.

3. **Vergain, Patrick.** Le blog de Patrick Vergain. [En línea] 2007. http://pvergain.files.wordpress.com/2007/10/framework35.jpg.

4. **Wikipedia, the free encyclopedia.** Common Language Infraestructura. [En línea] 26 de Noviembre de 2001.

5. **Trupin, Joshua.** Sharp new language: C# offers the power of C++ and the simplicity of Visual Basic. [En línea] Septiembre de 2000. http://msdn.microsoft.com/en-us/magazine/cc301505.aspx.

6. **ITU-T X.500.** The directory - Overview of Concepts, Models and Services. [En línea] 1988. http://dret.net/biblio/reference/x500.

7. **ISO 9594.** Information Processing Systems -- Open Systems Interconnection. [En línea] 1988.

8. **Wikepedia, the free encyclopedia.** LDAP. [En línea] 30 de Enero de 2005. http://es.wikipedia.org/wiki/LDAP.

9. **Microsoft Corporation.** Introduction to Windows Server 2003 Active Directory Application Mode. [En línea] 2002. http://www.microsoft.com/windowsserver2003/techinfo/overview/adam.mspx.

10. **RSA Laboratories.** [En línea] www.rsa.com.

11. —. Public-key Cryptography Standards (PKCS). [En línea] 1991. http://www.rsa.com/rsalabs/node.asp?id=2124.

12. **Wikipedia, the free encyclopedia.** PKCS. [En línea] 17 de Abril de 2006.

13. **RSA Laboratories.** PKCS #11 v2.20: Cryptographic Token Interface Standard. [En línea] 28 de Junio de 2004. ftp://ftp.rsasecurity.com/pub/pkcs/pkcs-11/v2-20/pkcs-11v2-20.doc.

14. —. PKCS #1 v2.1: RSA Cryptography Standard. [En línea] 14 de Junio de 2002. ftp://ftp.rsasecurity.com/pub/pkcs/pkcs-1/pkcs-1v2-1.pdf.

15. **Wikipedia, the free encyclopedia.** RSA. [En línea] 7 de Septiembre de 2003. http://es.wikipedia.org/wiki/Claves_RSA.

16. **Via.** Features of PadLock. [En línea]
http://www.via.com.tw/en/images/initiatives/padlock/fig_rsa.gif.

17. **Muppet Labs.** Prime Number Hide-and-Seek: How the RSA Cipher Works. [En línea]
http://www.muppetlabs.com/~breadbox/txt/rsa.html.

18. **Rijmen, J. Daemen y V.** AES Proposal: Rijndael, AES Algorithm Submission. 1999.

19. **Federal Information Processing Standards Publication 197.** ADVANCED ENCRYPTION STANDARD (AES). [En línea] 26 de Noviembre de 2001.
http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf.

20. **Schaad, J.** RFC 3565: Use of the Advanced Encryption Standard (AES) Encryption Algorithm in Cryptographic Message Syntax (CMS). [En línea] Julio de 2003.
http://www.ietf.org/rfc/rfc3565.txt.

21. **Wikipedia, the free encyclopedia.** SHA hash functions. [En línea] 21 de Septiembre de 2001. http://en.wikipedia.org/wiki/SHA.

22. **Clipperz.** [En línea] http://www.clipperz.com/files/clipperz.com/sha256-2.jpg.

23. **D. Eastlake, P. Jones.** US Secure Hash Algorithm. [En línea] Septiembre de 2001.
http://www.faqs.org/rfcs/rfc3174.html.

24. **Wikipedia, the free encyclopedia.** Encrypting File System. [En línea] 1 de Mayo de 2004.
http://en.wikipedia.org/wiki/File:EFSOperation.svg.